LabWindows/CVI

LabWindows/CVI User Interface Reference Manual

NATIONAL INSTRUMENTS™
The Software is the Instrument™

February 1998 Edition
Part Number 320683D-01

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

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

Fax-on-Demand Support
512 418 1111

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

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

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

Important Information

Warranty

Copyright

Trademarks

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

Contents

Chapter 3
Programming with the User Interface Library

Tables

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.

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 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.
- 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:

<>	Angle brackets enclose the name of a key on the keyboard—for example, <Shift>.
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options» Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** options from the last dialog box.
☞	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
bold	Bold text denotes the names of menus, menu items, parameters, and dialog box buttons.
bold italic	Notes appear in bold, italic text.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the appropriate word or value.

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

Customer Communication

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

1 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.

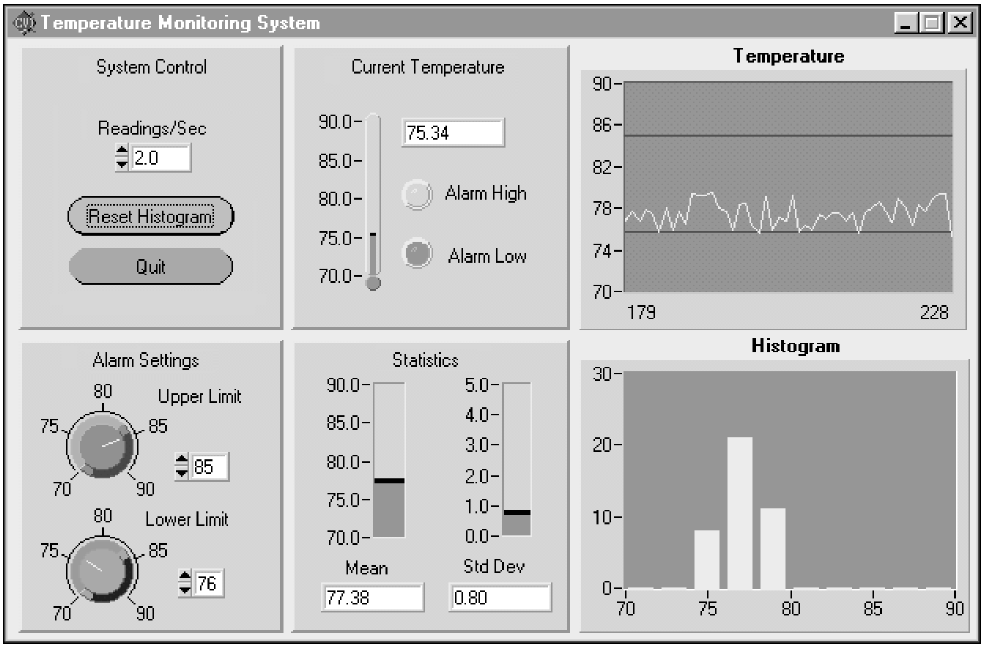


Figure 1-1. Typical LabWindows/CVI Graphical User Interface

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 LabWindows/CVI then passes to your C program. Actually, LabWindows/CVI controls generate multiple user interface events. For example, a single mouse-click on a command button can 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 makes it the active control. When a control receives 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 generated by each control type.

Controlling Your User Interface Using Callbacks or GetUserEvent

After you design your GUI in LabWindows/CVI, you must develop a C program to process the events generated from the user interface and to 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 might have a function in your program called `AcquireData` which you assign to a button labeled **Acquire**. Whenever the **Acquire** button is clicked, LabWindows/CVI passes all of the event information generated by the button directly to `AcquireData`, where you take appropriate action.

Alternatively, you can use an event-loop to process your user interface commit events in LabWindows/CVI. You poll for commit events by calling `GetUserEvent`. To process events other than commit events, you must install callback functions.

You can use either method in your program, or combine methods for added flexibility. In general, avoid using event-loops except when processing events from modal dialog boxes (also called pop-up panels). For more 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 your program can properly access it. Each control is given a *constant name* in the User Interface Editor. You use the constant names in calls to the User Interface Library functions 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 `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 Events

Value changed events are generated when the user of the GUI modifies the setting on a control, such as dragging the slider on a slide control with the mouse, or entering a character in a string control. Commit events are generated when the user of the GUI actually commits to an operation such as making a menu selection, typing in a number and pressing <Enter>, or releasing the mouse button after dragging the slider on a slide control. When you create a control you can assign one of the following control modes, which determine how the control generates events and to what extent the user can operate it:

- Normal
- Indicator
- Hot (the default)
- Validate

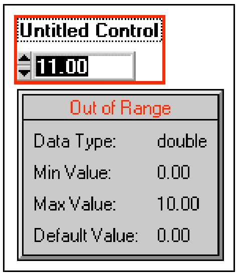
Normal specifies that the user can operate the control and that the control generates all events except commit events.

Indicator specifies that users cannot operate the control and that the control 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 a user acts upon it. Normally, a hot control generates a commit event when its state is changed. For example, if the user drags a binary switch from off to on and releases the mouse button, a commit event is generated. The following control types have unique rules on how they generate commit events:

- Hot numeric, string, and text box controls generate a commit event 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.
- Hot selection list controls not in check mode generate a commit event when the user presses <Enter> while the control is active, or double-clicks on a list item.
- Hot selection list controls in check mode generate a commit event when the user presses the spacebar while the control is active, or double-clicks on a list item.
- Hot graph controls generate a commit event 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.



Data Type:	double
Min Value:	0.00
Max Value:	10.00
Default Value:	0.00

Figure 1-2. Numeric Control with Out Of Range Message Pop-Up

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, and overlap multiple panels. Panels can contain other panels, called *child* panels. An example of a child panel within a parent panel appears 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 might be partially or completely hidden in the shrunken panel view.

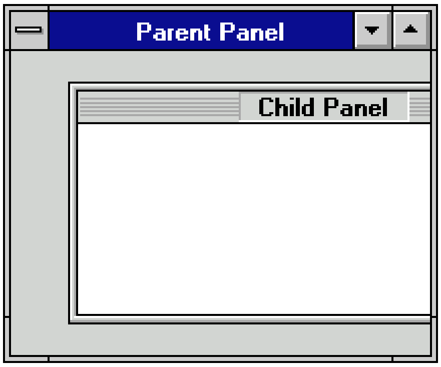


Figure 1-3. Child Panel within a Parent Panel

You can operate a panel from the keyboard in the following ways:

- 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.

You can operate a panel with the mouse in the following ways:

- 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.

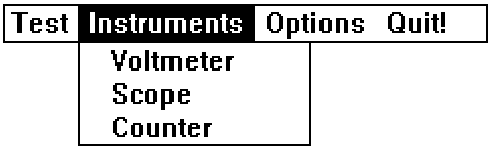


Figure 1-4. Menu Bar and Pull-Down Menu

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 appear to the side of submenu titles. A sample submenu appears in Figure 1-5.

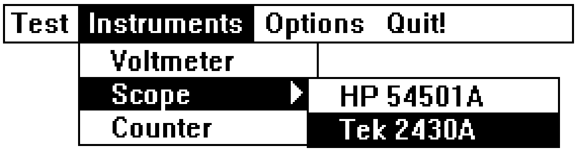


Figure 1-5. Pull-Down Menu with a Submenu

You can operate a menu bar from the keyboard in the following ways:

- Press <Alt> and the underlined letter of the menu title to display a menu or execute an immediate action menu. When a menu appears, 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.

You can operate a menu bar with the mouse in the following ways:

- Click on the menu title to display a menu or execute an immediate menu.
- Click on the menu item to execute an item within a menu. The menu disappears.
- Click anywhere outside the menu to remove the menu without executing the menu item.

You cannot select menu titles and menu items that are dimmed.

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 accessible menu bars contain a menu with the same underlined letter. On some controls, such as command buttons, pressing <Alt> and the underlined letter generates a commit event.

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:

```
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

You use *numeric controls* to input or view numeric values, for example, to input or display a voltage value. Numeric controls have many different graphical representations, including knobs, meters, thermometers, gauges, and dials. The numeric controls appear in Figure 1-6.

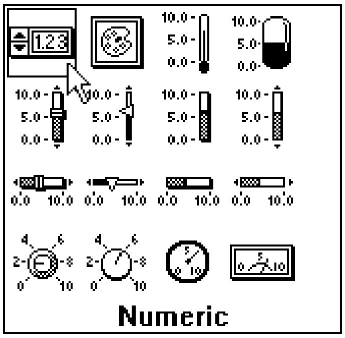


Figure 1-6. Numeric Controls

When a numeric control is not restricted to indicator mode, you can change its value from the keyboard or with the mouse.

You can operate a numeric control from the keyboard in the following ways:

- 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.

You can operate a numeric control with the mouse in the following ways:

- 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 you press the up or down arrow keys, when you drag the needle or slider and then release the mouse button, or when the you change the digital display of a numeric control and then press the <Enter> or <Tab> key or click on another object with the mouse.

String Controls

You use *string controls* to input or view strings of character values, such as a person's name. A string control appears in Figure 1-7.

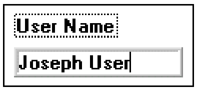


Figure 1-7. String Control

You can operate a string control from the keyboard in the following ways:

- 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.

Select text by holding down <Shift> and pressing <Home>, <End>, or the left or right arrow keys.

If the control mode of the string is hot or validate, a commit event is generated when you change the string and then press <Enter> or <Tab> or click on another object with the mouse.

Text Messages

Text messages are indicator controls you use 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 appears in Figure 1-8.

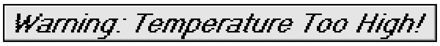
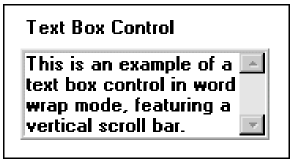

Warning: Temperature Too High!

Figure 1-8. 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

This is an example of a text box control in word wrap mode, featuring a vertical scroll bar.

Figure 1-9. Text Box Control

You can use the following keystrokes when you enter 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 you change the contents of the text box and then press the <Enter> or <Tab> key or click on another object with the mouse.

Command Button Controls

You use *command buttons* to trigger an action. You can give a button a label that corresponds to its action. The command buttons appear in Figure 1-10.

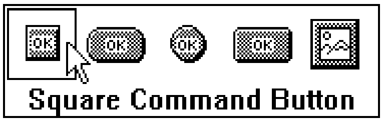


Figure 1-10. Command Button Controls

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

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

You use *toggle buttons* 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 not pressed its value is `0`. The toggle buttons appear in Figure 1-11.

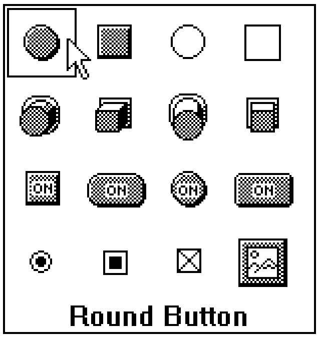


Figure 1-11. Toggle Button Controls

You can operate a toggle button from the keyboard in the following ways:

- 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 <spacebar> 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 you change 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. You operate LED controls like toggle button controls. The LED controls appear in Figure 1-12.

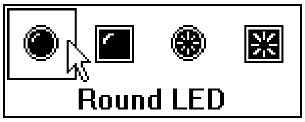


Figure 1-12. LED Controls

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. You operate binary switch controls like toggle button controls. The binary switches appear in Figure 1-13.

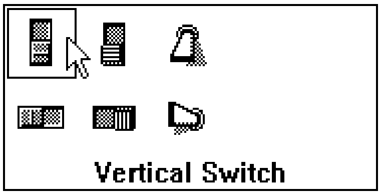


Figure 1-13. Binary Switch Controls

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 appear in Figure 1-14.

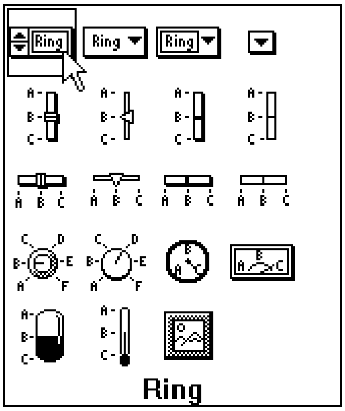


Figure 1-14. Ring Controls

You can operate a ring control from the keyboard in the following ways:

- 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 displays. A sample ring control in pop-up format appears in Figure 1-15.

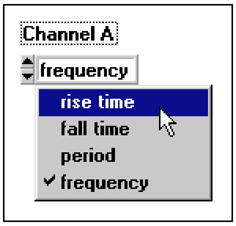


Figure 1-15. Ring Control in Pop-Up Format

You can operate a ring control pop-up from the keyboard in the following ways:

- Press <spacebar> 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.

You can operate a ring with the mouse as follows: if the ring control has arrows, click on the arrows to select the ring control items.

You can operate a ring control pop-up with the mouse in the following ways:

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

Note ***If the ring pop-up exceeds the size of the screen, you see a list box instead of a ring pop-up. Refer to the List Box Controls section 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 you change the ring's value.

List Box Controls

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

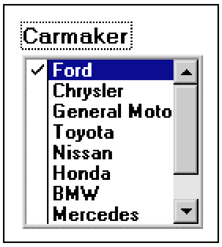


Figure 1-16. Selection List Control in Check Mode

The scroll bar on the right side of the list scrolls the list up and down.

You can operate a list box control from the keyboard in the following ways:

- 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 <spacebar> or <Enter> to toggle the check mark of the current item.

You can operate a list box control with the mouse in the following ways:

- 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.

You can generate a commit event when a list box is set to hot or validate mode in the following ways:

- When the control is set to **Check Mode**, select an item in the list box and press <spacebar> 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 can 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 appear in Figure 1-17.

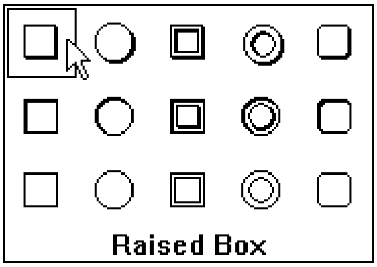


Figure 1-17. GUI Decorations

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 appears in Figure 1-18.

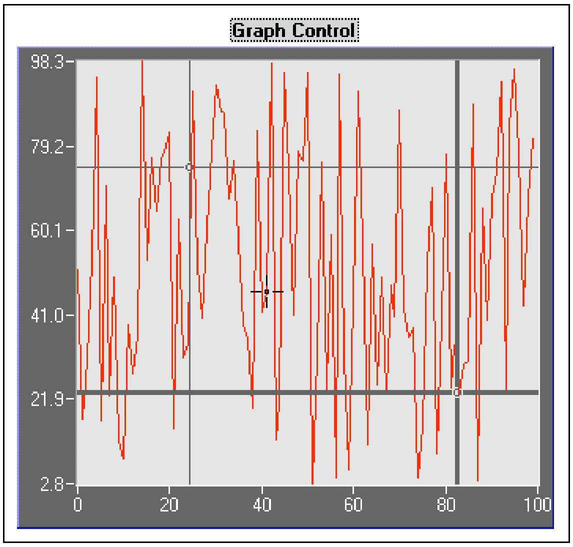


Figure 1-18. Graph Control

Graph controls can have one or more cursors associated with them. With cursors you can select a point or region of the graph for more processing or analysis. If the graph control is hot, cursors generate commit events. If you want to use cursors in a graph control, the mode of the graph control can be Normal, Hot, or Validate, but not Indicator.

You can use the keyboard to operate a graph with cursors, both free-form and snap-to-point cursors, as described in the following table.

Table 1-1. Keys for Cursor Operations

Type of Cursor	When you press...	You select...
Free-form and snap-to-point	<Page-Up>	Previous cursor.
	<Page-Down>	Next cursor.
Free-form	Left arrow key	Left 10 pixels.
	Right arrow key	Right 10 pixels.
	Up arrow key	Up 10 pixels.
	Down arrow key	Down 10 pixels.
	<Shift>-left arrow key	Left 1 pixel.
	<Shift>-right arrow key	Right 1 pixel.
	<Shift>-up arrow key	Up 1 pixel.
	<Shift>-down arrow key	Down 1 pixel.
	<Ctrl>-left arrow key	To the left edge of the plot area.
	<Ctrl>-right arrow key	To the right edge of the plot area.
	<Ctrl>-up arrow key	To the top edge of the plot area.
	<Ctrl>-down arrow key	To the bottom edge of the plot area.
	<Home>	To the lower left corner of the plot area.
	<End>	To the upper right corner of the plot area.

Table 1-1. Keys for Cursor Operations (Continued)

Type of Cursor	When you press...	You select...
Snap-to-point	Left arrow key	To the previous point on the current plot.
	Right arrow key	To the next point on the current plot.
	Up arrow key	To the next point on the current plot.
	Down arrow key	To the previous point on the current plot.
	<Shift>-left arrow key	Back 10 points on the current plot.
	<Shift>-right arrow key	Forward 10 points on the current plot.
	<Shift>-up arrow key	Forward 10 points on the current plot.
	<Shift>-down arrow key	Back 10 points on the current plot.
	<Ctrl>-left arrow key	Left to the closest point in the x direction on the current plot.
	<Ctrl>-right arrow key	Right to the closest point in the x direction on the current plot.
	<Ctrl>-up arrow key	Up to the closest point in the y direction on the current plot.
	<Ctrl>-down arrow key	Down to the closest point in the y direction on the current plot.
	<Home>	To the first visible point on the current plot.
	<End>	To the last visible point on the current plot.
	<Shift-Page Up>	To the previous plot.
	<Shift-Page Down>	To the next plot.

If you configure the graph as a hot control, an event is generated whenever you press one of the preceding keys.

You can operate a graph with cursors using the mouse in the following ways:

- Drag a cursor to move it. If the cursor is in snap-to-point mode, the cursor tracks the mouse until you release the mouse button, and then the cursor snaps to the nearest data point. If the cursor is in free-form mode, the cursor tracks the mouse until you release the mouse button, and then the cursor 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 you configure the graph control 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. If you want to enable zooming and panning in a graph control, the mode of the control can be Normal, Hot, or Validate, but not Indicator.

To start zooming in on a point, press the <Ctrl> key and left mouse button down over the point; at this point you can release the <Ctrl> key. The resolution in the viewport is continuously increased until you release the mouse. 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 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 scrolls 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 autoscaling in the graph, the autoscaling is disabled while you zoom or pan. If you plot additional data during zooming or panning, the zooming or panning terminates and the new data appears using autoscaling.

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 Figure 1-19.

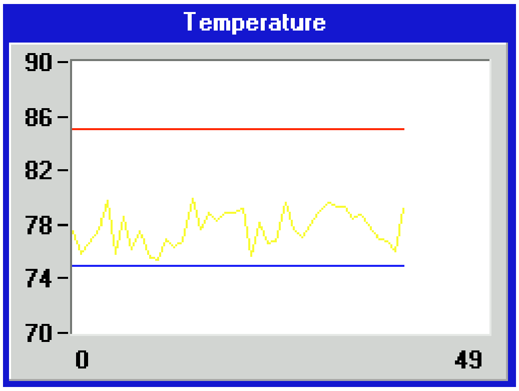


Figure 1-19. Strip Chart Control

Picture Controls

You use a picture control 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 connect a unit for testing. LabWindows/CVI also provides special types of command buttons, toggle buttons, and ring controls that can contain images. 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 1-2. Image Formats

Image Format	Platform
`.pcx`	Windows and UNIX
`.bmp`, `.dib`, `.rle`, `.ico`	Windows only
`.xwd`	UNIX only
`.wfm`	Windows 95/NT only

Timer Controls

You 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 appears in Figure 1-20.

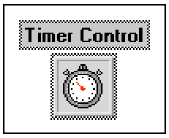


Figure 1-20. Timer Control

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

You use *canvas controls* as an arbitrary drawing surface. You can draw text, shapes, and bitmap images. LabWindows/CVI maintains an off-screen bitmap so that it can restore the appearance of the canvas 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

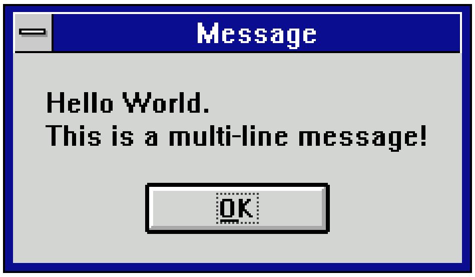
A *pop-up panel* is a panel that pops up, accepts user input, and then disappears. Another term for pop-up panel is *modal dialog box*. Pop-up panels can be stacked, with each new pop-up panel appearing on top of the previous one. When a pop-up panel is active it appears in the foreground and is the 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 multiline messages. Use the newline character (\n) to start a new line of text. A sample message pop-up panel appears in Figure 1-21.

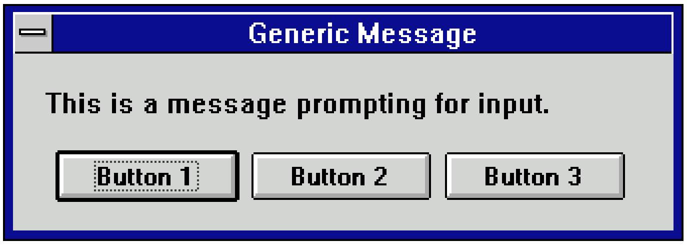


Hello World.
This is a multi-line message!

Figure 1-21. 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.

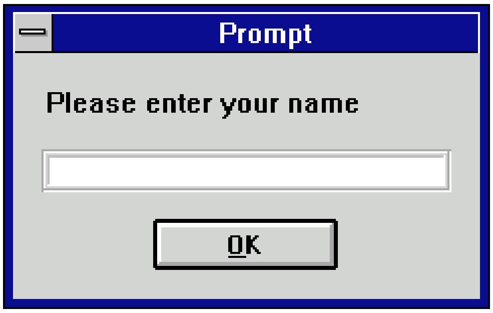


This is a message prompting for input.

Figure 1-22. 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.

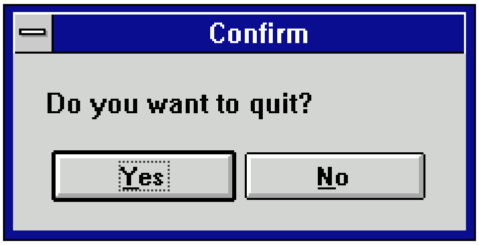


Please enter your name

Figure 1-23. 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.



Do you want to quit?

Figure 1-24. 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 Figure 1-25.

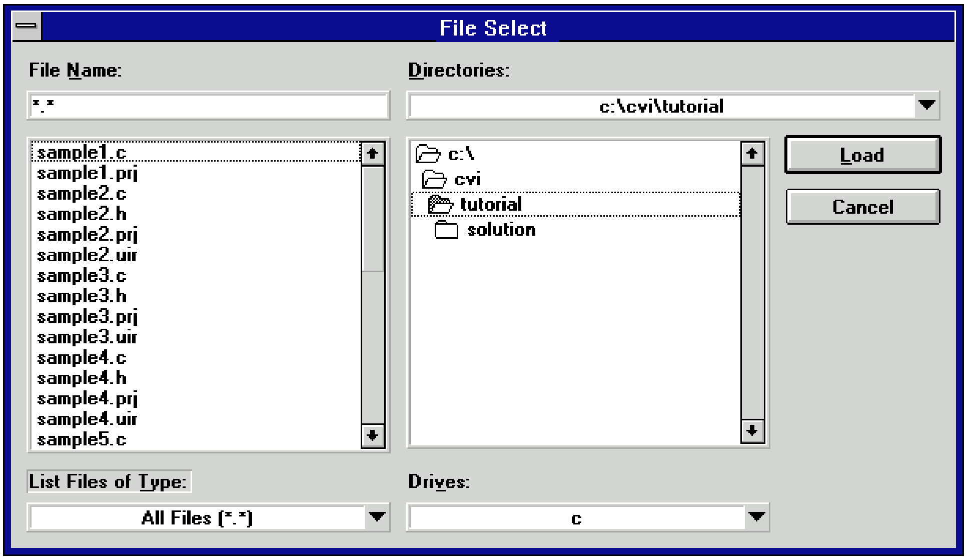


Figure 1-25. File Select Pop-Up Panel under Windows 3.1

A unique feature of this dialog box is the Directories ring control 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/NT appears in Figure 1-26.

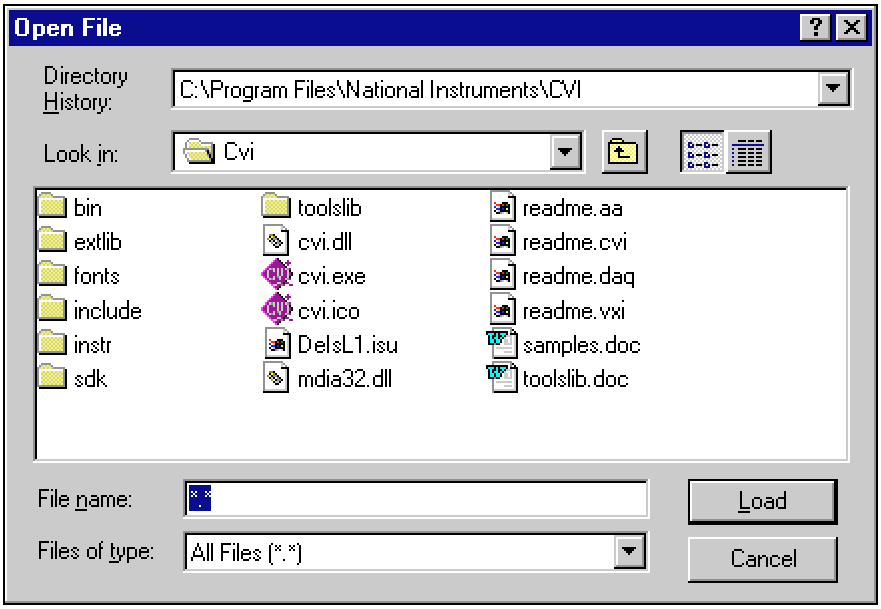


Figure 1-26. File Select Pop-Up Panel under Windows 95/NT

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 Figure 1-27.

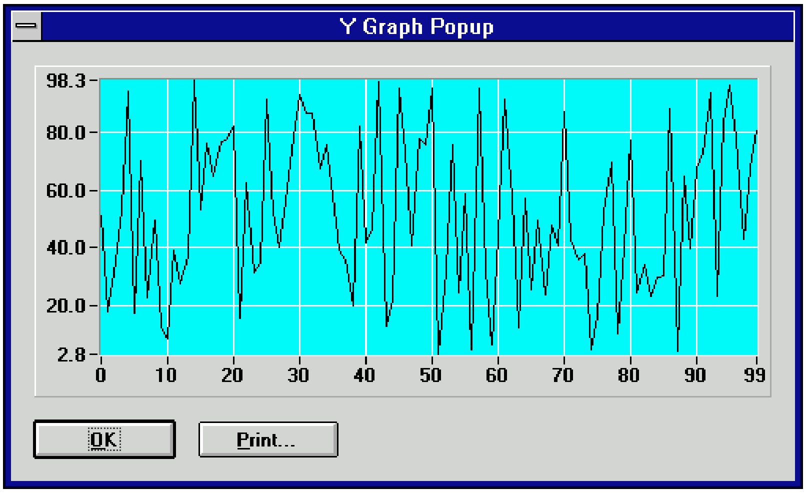


Figure 1-27. Graph Pop-Up Panel

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_SYSTEM_META_FONT
```

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 the **User Interface (*.uir)** menu item.

User Interface Editor Overview

A User Interface Editor window appears in Figure 2-1.

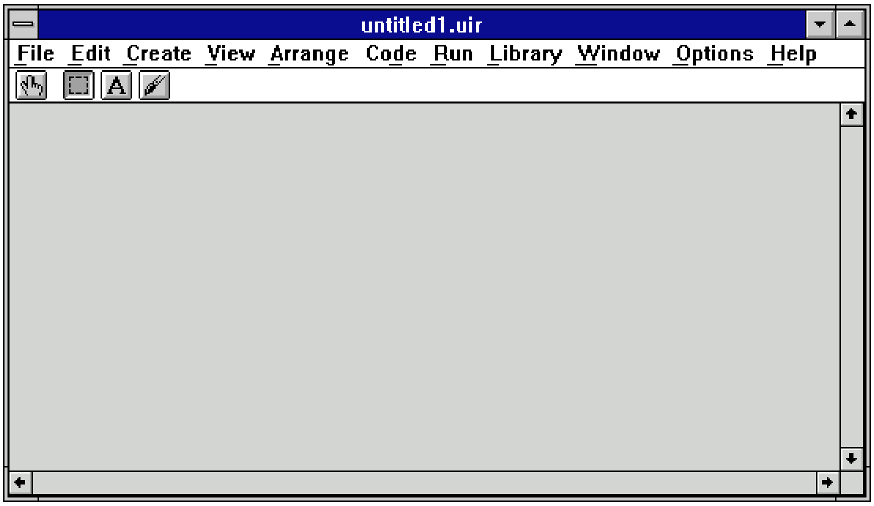

untitled1.uir
File Edit Create View Arrange Code Run Library Window Options Help

Figure 2-1. User Interface Editor Window

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 Pop-Up Menus of the User Interface Editor

You can open 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.

CodeBuilder 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 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, a `main` function must exist. If you lack the `main` function or any of the callback functions you reference in the `.uir` file, the code is incomplete.

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

Also with CodeBuilder, you can make sure that your automatically generated program terminates properly. For a CodeBuilder program to terminate successfully, you must include a call to `QuitUserInterface`. When you choose **Code»Generate»All Code,** the Generate All Code dialog box prompts you to choose which callback functions terminate the program. You can select one or more callback functions to ensure proper program termination.

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 appears in Figure 2-2.

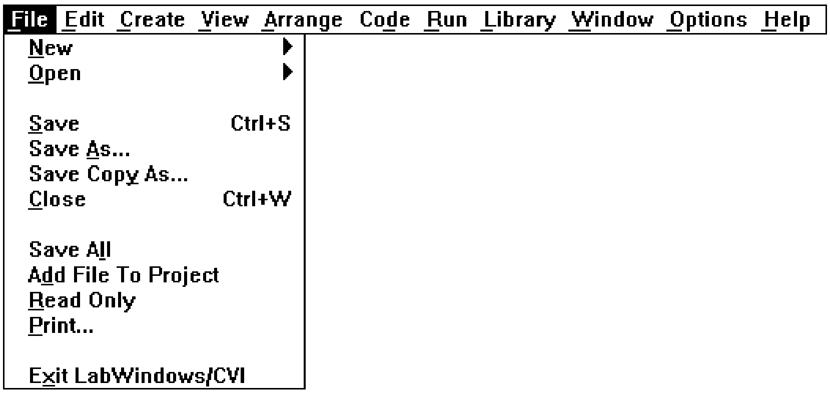


Figure 2-2. File Menu

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, 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. When you initially open a file, the **Read Only** command is disabled unless the file is read-only on disk.

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 correspond to the print attributes that 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.

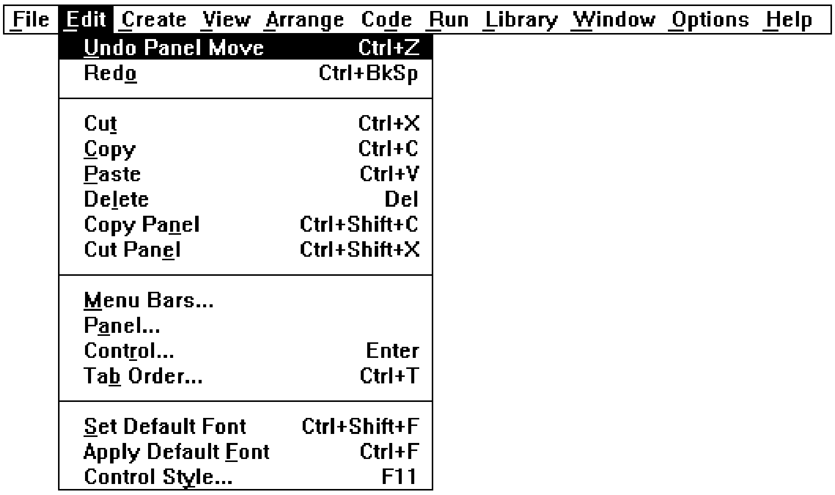


Figure 2-3. Edit Menu

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 can store up to 100 edit actions. You set the size of the undo stack 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 you can undo and redo appear dynamically 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 the previous contents of 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 the previous contents of 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 opens the Menu Bar List dialog box shown in Figure 2-4.

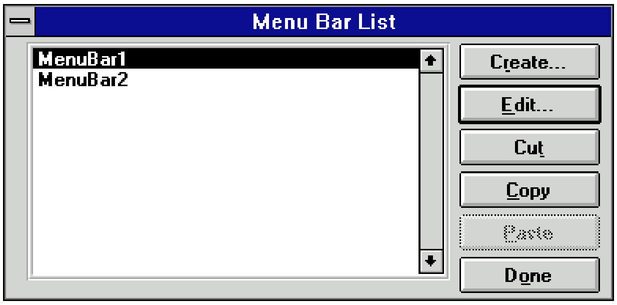


Figure 2-4. Menu Bar List Dialog Box

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

- **Create** opens a new Edit Menu Bar dialog box (Figure 2-5). After you create a menu bar, it appears below the currently selected menu bar in the menu bar list.
- **Edit** opens the Edit Menu Bar dialog box for the selected menu bar.
- **Cut** deletes the currently highlighted item in the menu bar list and copies it to the menu bar clipboard.
- **Copy** copies the currently highlighted item in the menu bar list to the menu bar clipboard.
- **Paste** inserts the contents of the menu bar clipboard to the menu bar list. 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 appears in Figure 2-5.

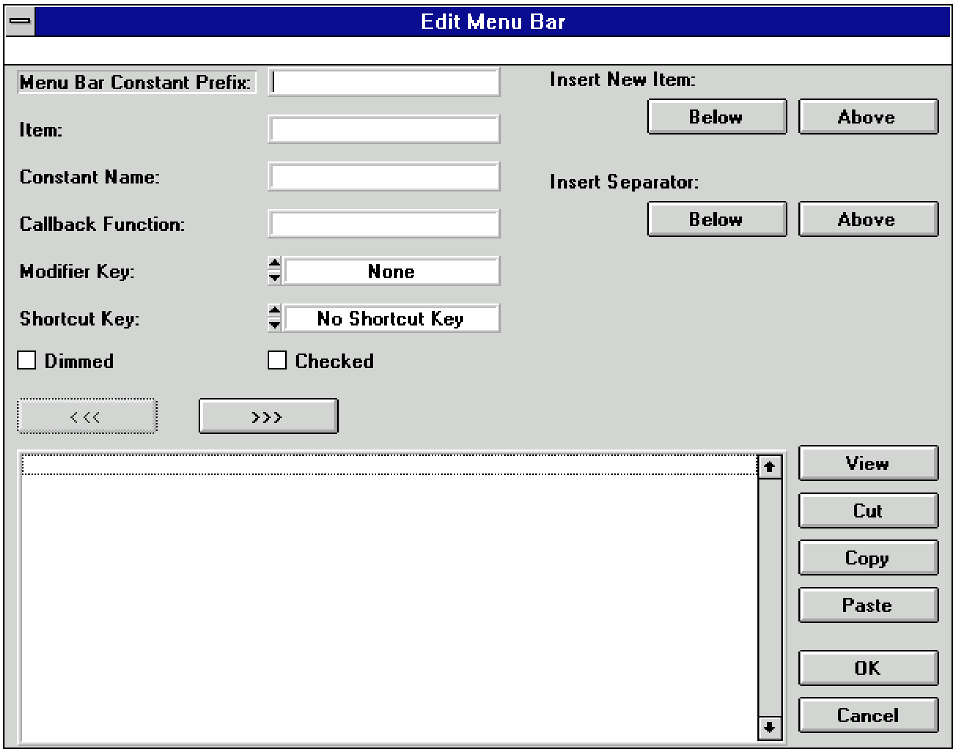
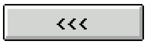


Figure 2-5. Edit Menu Bar Dialog Box

The Edit Menu Bar dialog box presents the following options:

- **Menu Bar Constant Prefix** is the resource ID for the menu bar. You pass this resource ID to `LoadMenuBar` to load the menu bar into memory. The Menu Bar Constant Prefix is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a Menu Bar Constant Prefix, the User Interface Editor assigns one for you when you save the `.uir` file.
- **Item** shows the name of the current menu, submenu, or menu command. If you type two underscores before any letter in the Item, the user can select the menu item by pressing <Alt> and that letter.
- **Constant Name** is appended to the Menu Bar Constant Prefix to form the ID for the current item. You pass the ID to functions such as `GetMenuBarAttribute` and `SetMenuBarAttribute`. `GetUserEvent` returns the ID when the current menu item generates a commit event.
- **Callback Function** is an optional field. In this box you can type the name of the function to be called when the current menu item generates an event.
- **Modifier Key** and **Shortcut Key** identify the keys that users can press to cause the current menu item to execute.
- **Dimmed** specifies whether or not the current menu item is initially dimmed.
- **Checked** specifies whether or not the current menu item initially has a check mark.
- **Insert New Item** inserts the next Item above or below the currently selected menu item.
- **Insert Separator** inserts a separator (a line) above or below the currently selected menu item.
- The left hierarchy button moves the currently selected menu item up one level in the submenu hierarchy.



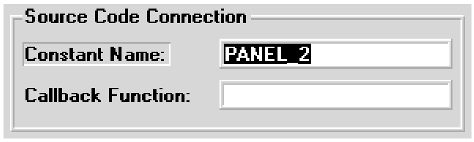
- The right hierarchy button moves the currently selected Item down one level in the submenu hierarchy.

- You click on **View** to display the current state of the menu bar and pull-down menus.

Panel

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

The Source Code Connection section of the Edit Panel dialog box appears in Figure 2-6.



Source Code Connection

Constant Name: PANEL_2

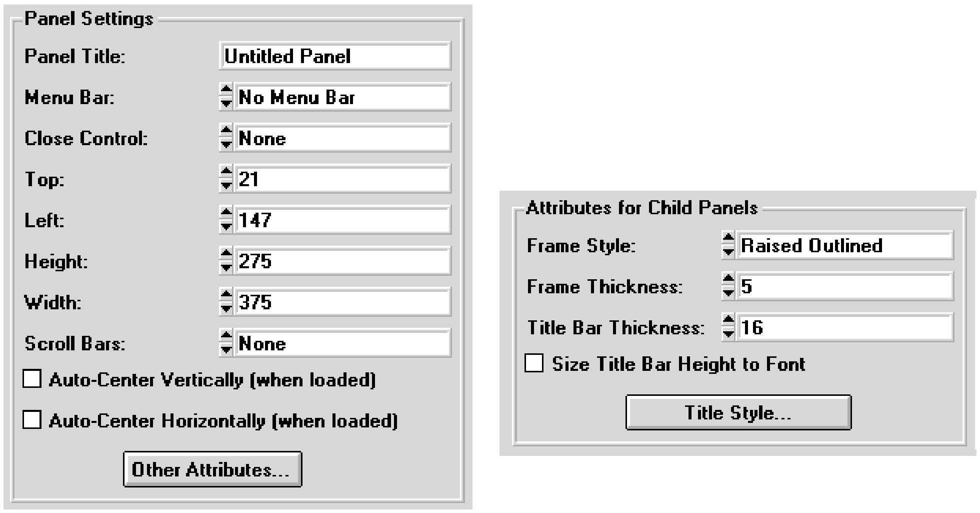
Callback Function:

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 `LoadPanel` to load the panel into memory. The Constant Name is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a Constant Name, the User Interface Editor assigns one when you save the `.uir` file.

In the Callback Function box you can type the name of the function to be called when an event is generated on the panel. Naming a callback function is optional.

The Panel Settings sections of the Edit Panel dialog box appear in Figure 2-7.



Panel Settings

Panel Title: Untitled Panel

Menu Bar: No Menu Bar

Close Control: None

Top: 21

Left: 147

Height: 275

Width: 375

Scroll Bars: None

☐ Auto-Center Vertically (when loaded)

☐ Auto-Center Horizontally (when loaded)

Attributes for Child Panels

Frame Style: Raised Outlined

Frame Thickness: 5

Title Bar Thickness: 16

☐ Size Title Bar Height to Font

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 LabWindows/CVI cannot implement some panel attributes under X Windows.***

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 Figure 2-8.

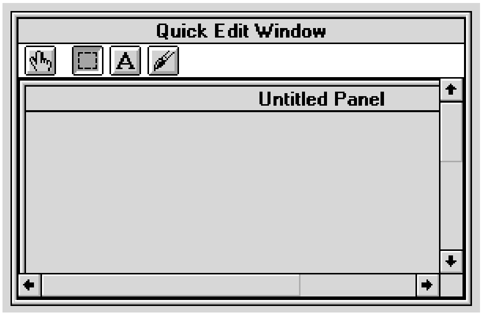


Figure 2-8. Quick Edit Window

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 more information, refer to the section *User Interface Editor Overview*, earlier in this chapter.

Control

The **Control** command opens the dialog box for editing the selected control. You can also double-click on a control to open this dialog box. The dialog box usually has five sections called *Source Code Connection*, *Control Settings*, *Control Appearance*, *Quick Edit Window*, and *Label Appearance*. The contents of the sections vary slightly depending on the type of control that you are editing. The Source Code Connection section of the control dialog box appears in Figure 2-9.

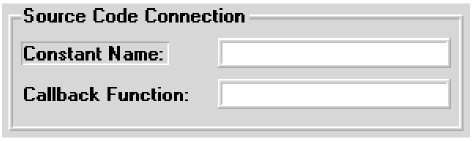
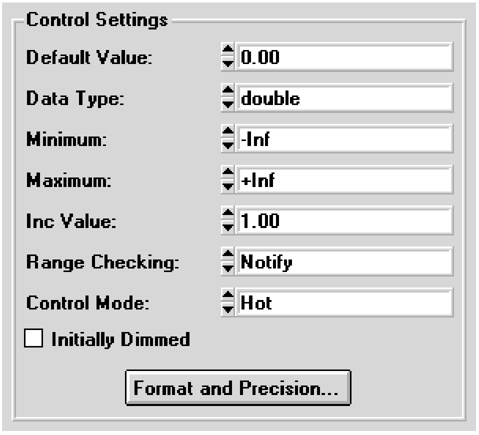


Figure 2-9. Source Code Connection

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 is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a Constant Name, the User Interface Editor assigns one for you when you save the `.uir` file.

In the Callback Function box you can type the name of the function to be called when an event is generated on the control. Naming a callback function is optional.

The Control Settings section of the dialog box displays specific attributes for the type of control that you are editing. It contains the data-specific attributes for the control. The Control Settings section for the Numeric control appears in Figure 2-10.

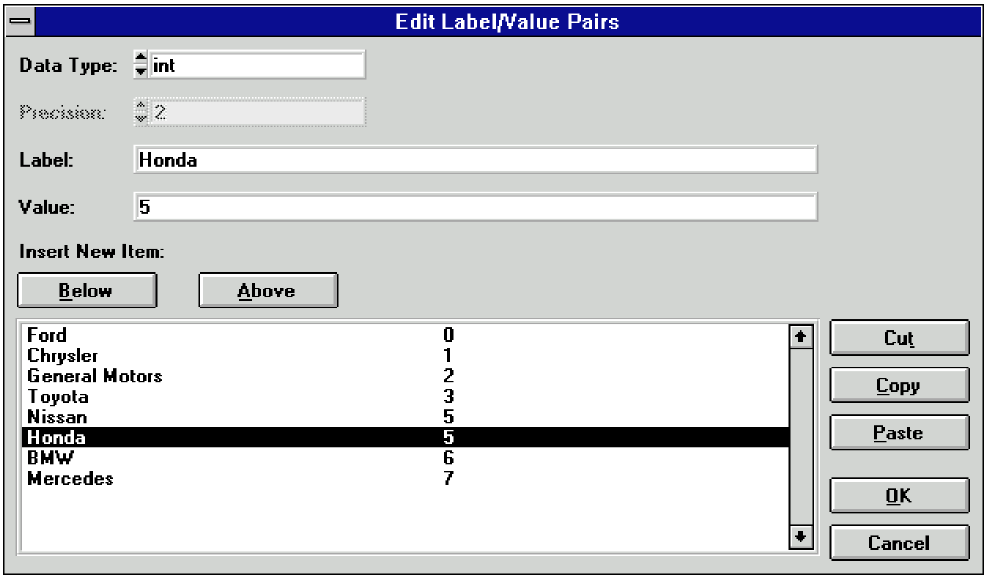

Control Settings

Default Value:	0.00
Data Type:	double
Minimum:	-Inf
Maximum:	+Inf
Inc Value:	1.00
Range Checking:	Notify
Control Mode:	Hot

☐ Initially Dimmed

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 Figure 2-11.



Data Type: int
Precision: 2
Label: Honda
Value: 5
Insert New Item:

Ford	0
Chrysler	1
General Motors	2
Toyota	3
Nissan	5
Honda	5
BMW	6
Mercedes	7

Figure 2-11. 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 displays specific attributes for the type of control that you are editing. It contains attributes pertaining to the physical appearance of the control. The Control Appearance section for the Numeric control appears in Figure 2-12.

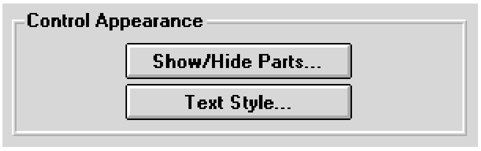


Figure 2-12. Control Appearance for a Numeric Control

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 appears in Figure 2-13.

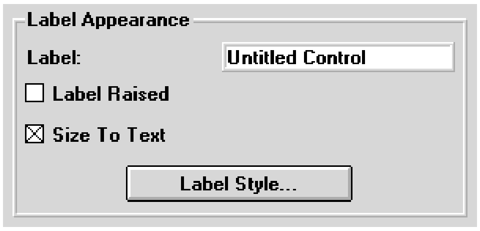


Figure 2-13. Label Appearance for a Numeric Control

If you type a double underscore before any letter in the label text, the letter appears underlined in the label. The user can select the control by pressing <Alt> and the underlined letter, provided that no accessible menu bars contain a menu with the same underlined letter.

The Quick Edit Window section of the Edit Control dialog box appears in Figure 2-14.

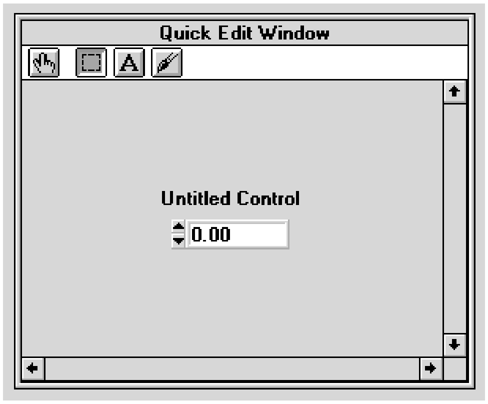


Figure 2-14. Quick Edit Window

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 immediate reflects any changes you make in other sections 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 positions itself at the end of the tab order. When you copy and paste a control, the tab position of the new control is immediately prior to the control you copied. Select **Tab Order** from the **Edit** menu to put the panel into tab order edit mode as shown in Figure 2-15.

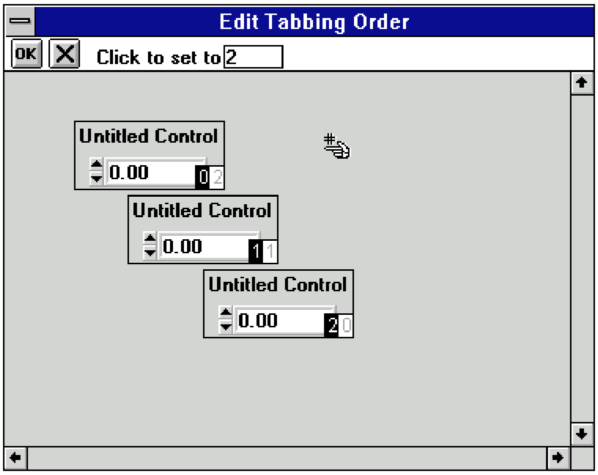


Figure 2-15. Edit Tab Order Dialog Box

Click on a control with the *special pointer cursor* to change the tab position of a control to the number in the Click to set to box.

You can change the cursor to the *special 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 the **OK** button accepts the new tab order.

Clicking on the *close button* erases the new tab order and restores 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.

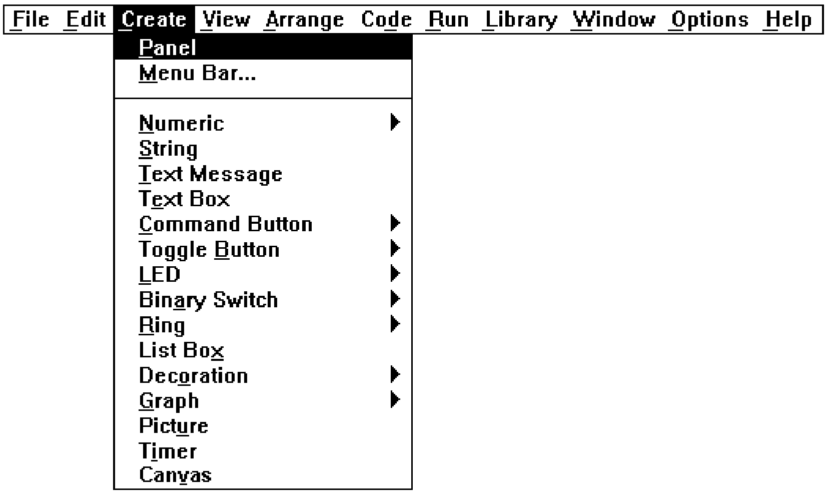


Figure 2-16. Create Menu

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 opens 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. After you create a control you can modify it using the items in the **Edit** menu. The *Edit Menu* section of this chapter provides details on editing controls.

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.

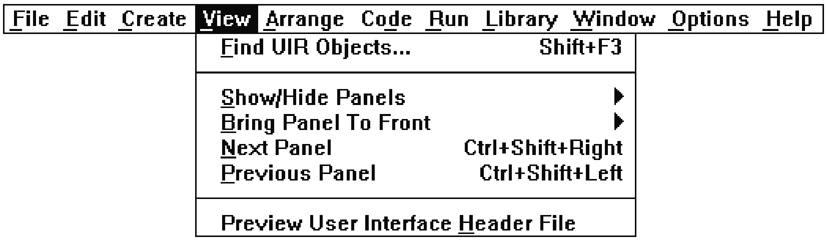
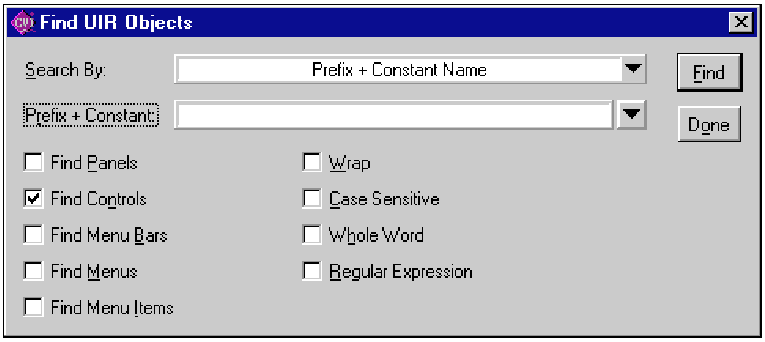

File Edit Create View Arrange Code Run Library Window Options Help
Find UIR Objects... Shift+F3
Show/Hide Panels
Bring Panel To Front
Next Panel Ctrl+Shift+Right
Previous Panel Ctrl+Shift+Left
Preview User Interface Header File

Figure 2-17. View Menu

Find UIR Objects. . .

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



Search By: Prefix + Constant Name

Prefix + Constant:

[✓] Find Controls

Figure 2-18. Find UIR Objects Dialog Box

Select the type or types 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.

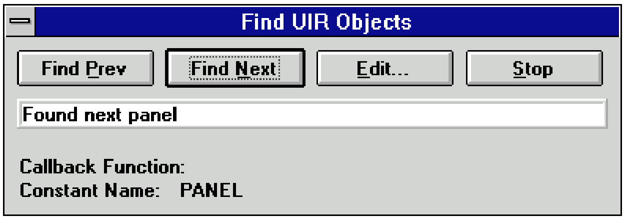
Wrap continues your search from the beginning of the file after you reach the end of the file.

Case Sensitive finds only instances of the specified text that match exactly.

Whole Word finds the specified text only when it is surrounded by spaces, punctuation marks, or other characters not part of a word. LabWindows/CVI treats the characters `A` through `Z`, `0` through `9`, and underscore (_) as parts of a word.

Regular Expression causes LabWindows/CVI to treat certain characters in the search string control as regular expression characters instead of literal characters. Table 4-1 of the *LabWindows/CVI User Manual* describes the regular expression characters.

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



Found next panel

Callback Function:
Constant Name: PANEL

Figure 2-19. Find UIR Objects Dialog Box after a Search Executes

This dialog box allows you to browse through the list of matches. As you come to each object, its callback function name and label appear, and the object is highlighted in the `.uir` file.

Find Prev searches backward for the previously matched object.

Find Next searches for the next matching object.

Edit terminates the search and opens the Edit dialog box for the user interface object currently highlighted.

Stop terminates the search.

Show/Hide Panels

The **Show/Hide Panels** command has a submenu, as shown in Figure 2-20.

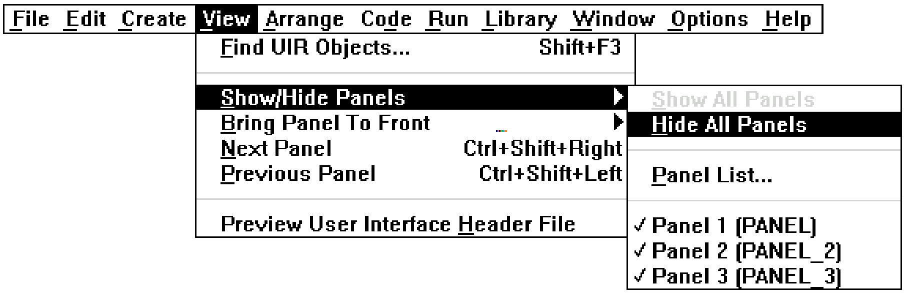

File Edit Create View Arrange Code Run Library Window Options Help
Find UIR Objects... Shift+F3
Show/Hide Panels
Bring Panel To Front
Next Panel Ctrl+Shift+Right
Previous Panel Ctrl+Shift+Left
Preview User Interface Header File
Show All Panels
Hide All Panels
Panel List...
✓ Panel 1 (PANEL)
✓ Panel 2 (PANEL_2)
✓ Panel 3 (PANEL_3)

Figure 2-20. Show/Hide Panel Submenu

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 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 opens a Source code window with a preview of the header file that LabWindows/CVI generates when you save the `.uir` file in the User Interface Editor window.

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.

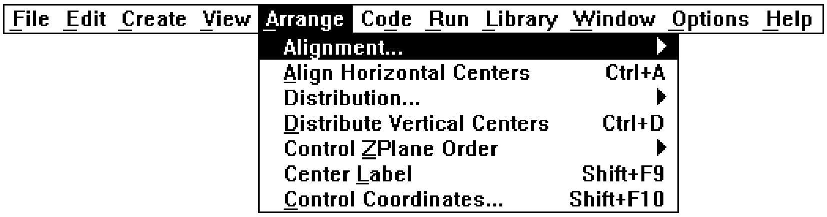

File Edit Create View Arrange Code Run Library Window Options Help
Alignment...
Align Horizontal Centers Ctrl+A
Distribution...
Distribute Vertical Centers Ctrl+D
Control ZPlane Order
Center Label Shift+F9
Control Coordinates... Shift+F10

Figure 2-21. Arrange Menu

Alignment

The **Alignment…** command allows you to align controls on a panel. You can 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 Figure 2-22.

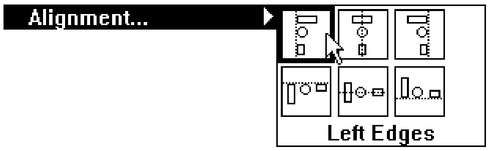


Figure 2-22. Alignment Menu

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** command performs the same action as the **Alignment…** command, using the option you 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 Figure 2-23.

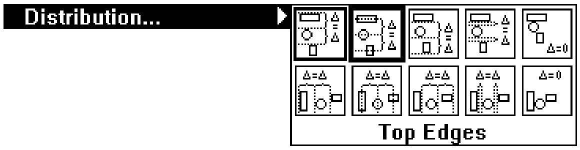


Figure 2-23. Distribution Submenu

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 you last selected in the **Distribution** command submenu.

Control ZPlane Order

The **Control ZPlane Order** option lets you sets the sequence in which overlapped controls are drawn. Controls are always drawn in order, from the back to the front of the z-plane order. The **Control ZPlane Order** submenus present 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 where you can 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.

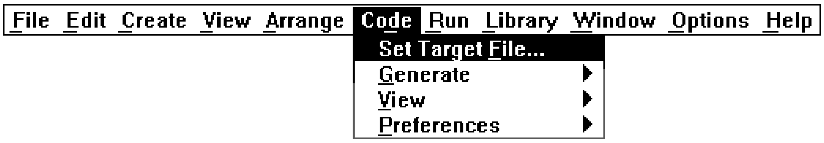

File Edit Create View Arrange Code Run Library Window Options Help
Set Target File...
Generate
View
Preferences

Figure 2-24. Code Menu

Set Target File

Use the **Set Target File…** command to specify to which file LabWindows/CVI generates code. Selecting this command opens the Set Target File dialog box, 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.

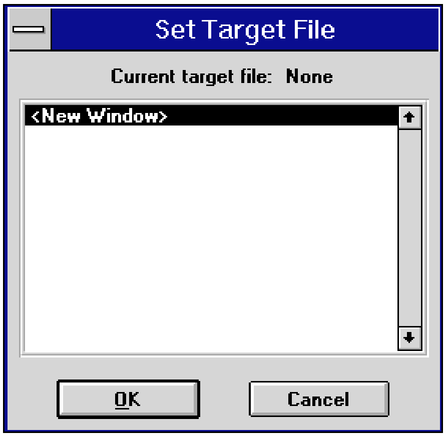


Figure 2-25. Set Target File Dialog Box

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

You access the CodeBuilder features of LabWindows/CVI in the **Generate** menu item. 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 you specify with the **Bracket Styles** command in the **Options** menu of the Source window for your project.

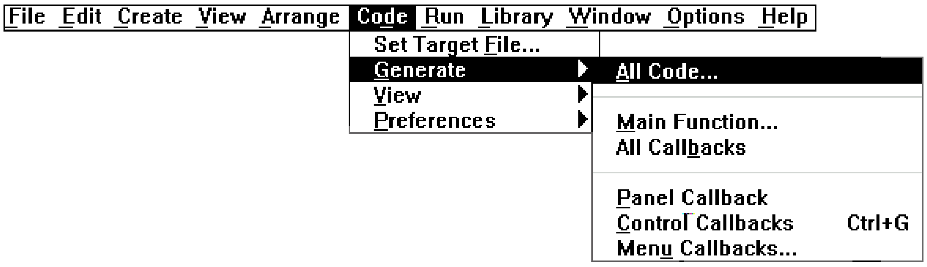


Figure 2-26. Generate Menu

The **Panel Callback** command is available if you specified a callback function for the currently active panel. The **Control Callback** command is available if you have specified callback functions for any of the currently selected controls. The **Menu Callbacks** command is available if you have a menu bar that contains items for which you specified a callback. The **All Callbacks** command is available when any of the **Panel Callback**, **Control Callbacks**, or **Menu Callbacks** commands are available.

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

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

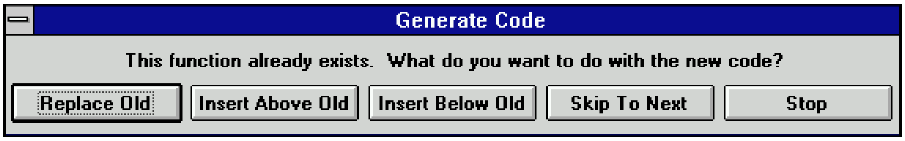

This function already exists. What do you want to do with the new code?

Figure 2-27. Generate Code Dialog Box

All Code

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

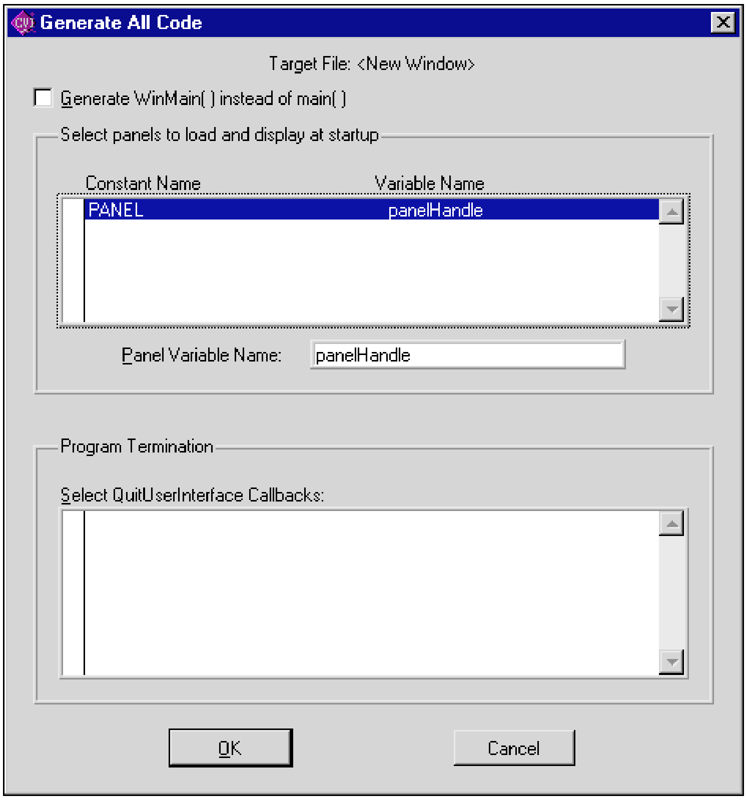

Generate All Code
Target File: <New Window>
Generate WinMain( ) instead of main( )
Select panels to load and display at startup
Constant Name
Variable Name
PANEL
panelHandle
Panel Variable Name:
panelHandle
Program Termination
Select QuitUserInterface Callbacks:
OK
Cancel

Figure 2-28. Generate All Code Dialog Box

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

Note ***Callback functions associated with close controls are automatically checked in the Program Termination section of the Generate All Code dialog box. You can define a control to be a close control 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 or function 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 functions you selected to terminate program execution include a call to the User Interface Library `QuitUserInterface` function.

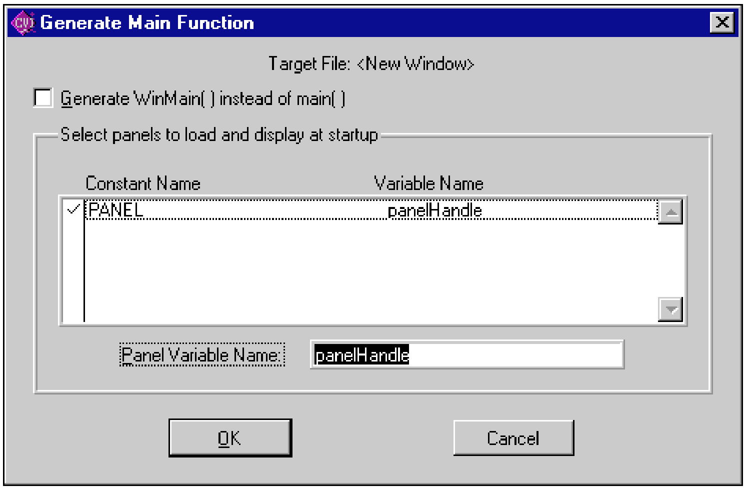
Unless you have selected the **Always Append Code to End** option, LabWindows/CVI places the skeleton code for each callback function at the cursor position in the target file. If the cursor is inside an existing function, LabWindows/CVI repositions the cursor at the end of that function before inserting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

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 opens the Generate Main Function dialog box, shown in Figure 2-29. This dialog box prompts you to choose the panels 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.



Target File: <New Window>

☐ Generate WinMain( ) instead of main( )

Select panels to load and display at startup

Constant Name	Variable Name
✓ PANEL	panelHandle

Panel Variable Name: panelHandle

Figure 2-29. Generate Main Function Dialog Box

Note *If you previously selected* **Code»Generate»All Code** *command, you do not have 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 or panels 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 **Generate WinMain( ) instead of main( )** checkbox 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, CodeBuilder generates a `DLLMain` function and places the bulk of the User Interface function calls in a function called `InitUIForDLL`. Call `InitUIForDLL` in your DLL at the point you want to load and display panels.

When you link your executable or DLL in an external compiler, you must include a call to `InitCVIRTE` in `WinMain`, `main`, or `DLLMain` (or `DLLEntryPoint` for Borland C/C++). In a DLL, you also must include a call to `CloseCVIRTE`. Refer to the *Calling InitCVIRTE and CloseCVIRTE* section in Chapter 3, *Windows 95/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 the callback function associated with a panel. 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 the callback functions associated with one or more controls. 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** opens 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.

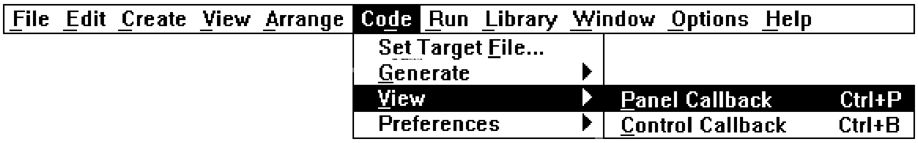


Figure 2-30. View Menu

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** menu.

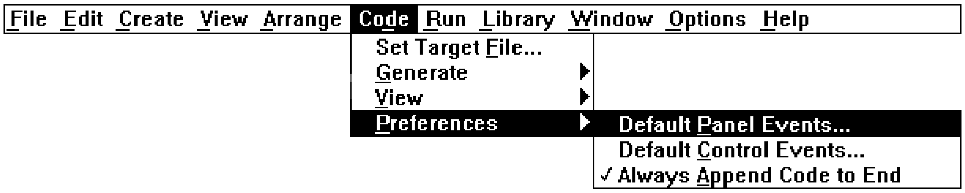

File Edit Create View Arrange Code Run Library Window Options Help
Set Target File...
Generate
View
Preferences
Default Panel Events...
Default Control Events...
✓ Always Append Code to End

Figure 2-31. Preferences Menu

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** opens the Panel Callback Events dialog box. Selecting **Code»Preferences»Default Control Events** opens the Control Callback Events dialog box.

To set the **Default Panel Events** or **Default Control Events**, select the events you want to 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, for which the 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:

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

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. Refer to 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. Refer to 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.

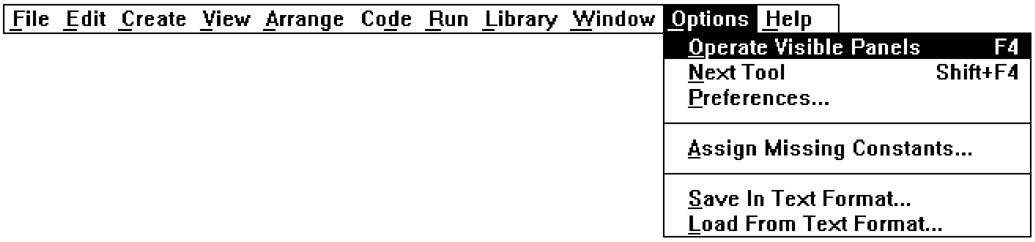

File Edit Create View Arrange Code Run Library Window Options Help
Operate Visible Panels F4
Next Tool Shift+F4
Preferences...
Assign Missing Constants...
Save In Text Format...
Load From Text Format...

Figure 2-32. 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 Operating tool, shown at left. 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

Coloring Mode

Preferences

Selecting **Preferences…** opens the User Interface Editor Preferences dialog box, as shown in Figure 2-33.

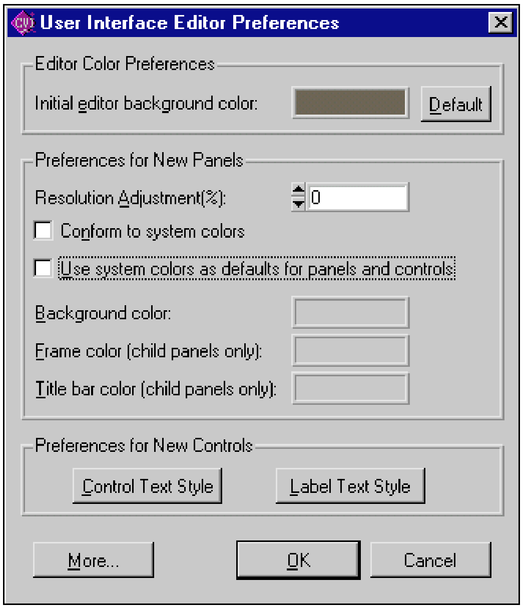

User Interface Editor Preferences
Editor Color Preferences
Initial editor background color:
Default
Preferences for New Panels
Resolution Adjustment(%):
0
Conform to system colors
Use system colors as defaults for panels and controls
Background color:
Frame color (child panels only):
Title bar color (child panels only):
Preferences for New Controls
Control Text Style
Label Text Style
More...
OK
Cancel

Figure 2-33. User Interface Preferences Dialog Box

Editor Color Preferences

Use the **Editor Color Preferences** section of the User Interface Editor Preferences dialog box to set the initial background color of User Interface Editor windows.

Preferences for New Panels

Use the **Preferences for New Panels** section of the User Interface Editor Preferences dialog box to set initial attribute values for each panel that you create in the User Interface Editor.

Use the **Resolution Adjustment** option to specify the degree to which LabWindows/CVI scales your panels and their contents when you display them on screens with resolutions different than the one on which you create them. This option also appears in the Other Attributes dialog box that you can activate from the Edit Panel dialog box. To programmatically override this setting you can call `SetSystemAttribute` with the `ATTR_RESOLUTION_ADJUSTMENT` attribute before calling `LoadPanel` or `LoadPanelEx`.

Use the **Conform to system colors** option to force panels and the controls they contain to use the system colors on Windows 95/NT. This option also appears in the Other Attributes dialog box that you can activate from the Edit Panel dialog box. To programmatically set this option you can call `SetPanelAttribute` with the `ATTR_CONFORM_TO_SYSTEM` attribute. When this option is enabled, you cannot change any panel or control colors.

When the **Use system colors as defaults for panels and controls** option is in effect, LabWindows/CVI uses the system colors as the initial colors for panels and controls you create on Windows 95/NT. You can subsequently change the colors without restriction.

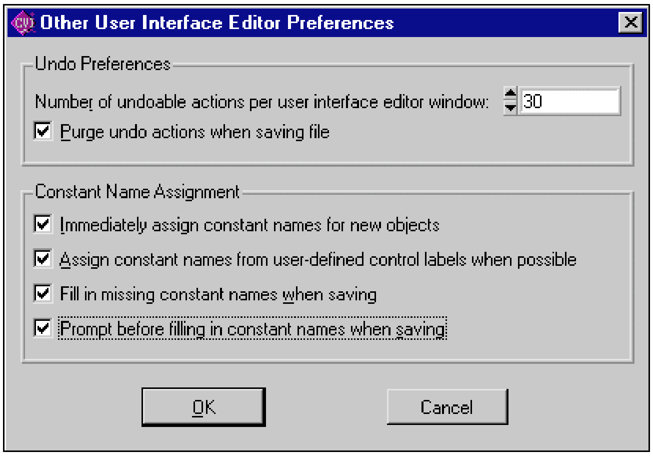
You must disable two options, **Conform to system colors** and **Use system colors as defaults for panels and controls**, in order to set the following options: **Background color**, **Frame color**, and **Titlebar color**. The Frame color and Titlebar color options have effect only when you load a panel as a child panel. To change each of these three options in the User Interface Editor you can use the Paintbrush tool on the background, frame, or titlebar of a panel. To set these colors programmatically, use `SetPanelAttribute` with the `ATTR_BACKCOLOR`, `ATTR_FRAME_COLOR`, and `ATTR_TITLE_BACKCOLOR` attributes.

Preferences for New Controls

Use the Preferences for New Controls section of the User Interface Editor Preferences dialog box to set initial attribute values for each control that you create in the User Interface Editor. The **Control Text Style** and **Label Text Style** command buttons allow you to select the initial font and text style for all new controls.

More

The **More** command button opens the Other UI Editor Preferences dialog box, as shown in Figure 2-34.


Other User Interface Editor Preferences

Undo Preferences

Number of undoable actions per user interface editor window: 30

[✓] Purge undo actions when saving file

Constant Name Assignment

[✓] Immediately assign constant names for new objects

[✓] Assign constant names from user-defined control labels when possible

[✓] Fill in missing constant names when saving

[✓] Prompt before filling in constant names when saving

Figure 2-34. Other UI Editor 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. If you want the Undo buffer to empty every time you save a file, select the option to **Purge undo actions when saving file**.

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 `.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 you do not have a chance to customize the control labels before the User Interface Editor generates the constant name. Consequently, the User Interface Editor bases the constant name on the control type.

If you choose to disable the **Immediately assign constant names for new objects** option, 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. You can call `LoadPanel` and `LoadPanelEx` on `.tui` files.

☞ **Note** ***If you have a large number of objects in your User Interface Editor Window, loading a*** `.tui` ***file can take significantly longer than loading a comparable*** `.uir` ***file.***

☞ **Note** ***The*** `.tui` ***file format in LabWindows/CVI 5.0 and later differs from previous versions. If you use*** `.tui` ***files to find differences between versions of your*** `.uir` ***files and you created*** `.tui` ***files in previous versions of LabWindows/CVI, create new baseline*** `.tui` ***files for your*** `.uir` ***files.***

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.

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 you can develop projects in LabWindows/CVI in many ways, you might 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.
4. Use CodeBuilder to create skeleton code for your source file. Refer to the *CodeBuilder Overview* section of Chapter 2, *User Interface Editor Reference*, for more information.
5. Edit the skeleton code that CodeBuilder generates. Add code to the callback functions so that they take the appropriate actions in response to user interface events.
6. Add the source code (`.c`), include (`.h`), and user interface (`.uir`) files to your project list and save the project (`.prj`) file.

If you edit the program and `.uir` file concurrently, you must recompile your program if you modify the `.uir` file before you run your program. 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 Compile Options dialog box of the Project window.

If you do not pass absolute pathnames for `.uir` files and image files to the `LoadMenuBar`, `LoadPanel`, or `DisplayImageFile` functions, the User Interface Library searches for them in the following places and order:

1. The project list.
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, do not use absolute path names in calls to 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*, two ways exist for creating Graphical User Interface (GUI) objects for your application program. You can create objects programmatically using function calls or interactively using the User Interface Editor.

After 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. In general, avoid using event loops except for processing events from pop-up panels.

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

Resource IDs and Panel and Menu Bar Handles

You can assign resource IDs to user interface objects in the User Interface Editor. You use the IDs in your program to identify the objects. For controls, menus, submenus, and menu items, you use the IDs throughout your program. For and panels and menu bars, you use IDs only to load the panels and menu bars from the `.uir` file. Thereafter, you use the panel and menu bar handles that `LoadPanel`, `LoadPanelEx`, `LoadMenuBar`, `LoadMenuBarEx` return.

When you create your GUI programmatically, `NewPanel` and `NewMenuBar` return handles, whereas `NewCtrl`, `NewMenu`, `NewSubMenu`, and `NewMenuItem` return IDs.

Panel and menu bar handles represent top-level objects that can contain other objects. For example, if you call `DiscardPanel` on a panel handle, LabWindows/CVI destroys the panel and all of the controls it contains. IDs, on the other hand, represent objects that must be contained within higher-level objects and that cannot contain other objects. For example, a control must be within a panel and cannot contain other controls.

If You Design Your GUI in the User Interface Editor...

If you design your GUI in the User Interface Editor, you assign callback functions and unique ID constant names to objects in the editor. LabWindows/CVI automatically saves their declarations in an include (`.h`) file whenever you save the resource file. You must include this `.h` 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. You obtain panel and menu bar handles at run time when you use the `LoadPanel` and `LoadMenuBar` functions.

Note ***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. Refer to the*** *CodeBuilder Overview* ***section of Chapter 2,*** *User Interface Editor Reference*, ***for more information.***

If You Build Your GUI Programmatically...

If you build you GUI programmatically, you obtain panel and menu bar handles from the `NewPanel` and `NewMenuBar` functions. You obtain IDs from 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 can 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 appended 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 is `SCOPE_POWER`. The application program uses the complete constant name to reference the control.

The following rules apply when you assign 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 can 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 can 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 `OPEN`, the complete constant name is `MAIN_FILE_OPEN`. 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 your program at event time.

Table 3-1. User Interface Events

Event Type	Event on the GUI	Information Passed to Program
Control and Menu Event	`EVENT_COMMIT`	Which panel or menu bar, which control or menu item.
Control Event	`EVENT_VAL_CHANGED`	Which panel, which control.
Control and Panel Event	`EVENT_LEFT_CLICK`	Which panel, which control, mouse y- and x-coordinates.
	`EVENT_LEFT_DOUBLE_CLICK`	Which panel, which control, mouse y- and x-coordinates.
	`EVENT_RIGHT_CLICK`	Which panel, which control, mouse y- and x-coordinates.
	`EVENT_RIGHT_DOUBLE_CLICK`	Which panel, which control, mouse y- and x-coordinates.
	`EVENT_KEYPRESS`	Which panel, which control, key code, pointer to key code.
	`EVENT_GOT_FOCUS`	Which panel, which control
	`EVENT_LOST_FOCUS`	Which panel, which control.
	`EVENT_DISCARD`	Which panel, which control.
Timer Control Event	`EVENT_TIMER_TICK`	Pointer to the current time (`double *`), pointer to time since the callback last received an `EVENT_TIMER_TICK` (`double *`).
Panel Event	`EVENT_CLOSE`	Which panel.
	`EVENT_PANEL_SIZE`	Which panel.
	`EVENT_PANEL_MOVE`	Which panel.
Main Callback Event	`EVENT_IDLE`	Obsolete. Use timer controls instead.
	`EVENT_END_TASK`	Windows only. Received when Windows wants to quit. Return a non-zero value to abort the termination.

Using Callback Functions to Respond to User Interface Events

Callback functions respond to all events listed in Table 3-1. The C prototypes for the callback functions are in `userint.h`. You can have callback functions 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. Refer to the discussion of `InstallMainCallback` in the *Special User Interface Functions* section of this chapter.

LabWindows/CVI passes event information from the GUI to your callback functions. For example, callback functions receive the type of user interface event that occurred, such as `EVENT_LEFT_CLICK`, 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 when responding to events.

☒ A top-level panel callback receives the `EVENT_CLOSE` message callback when the user executes the **Close** command from the **System** menu or clicks on the Close button in the upper right corner of the panel title bar.

A panel callback receives the `EVENT_SIZE` and `EVENT_MOVE` messages when the user resizes or moves the panel. The panel callback does not receive these messages when you programmatically resize or move a panel.

The main callback receives the `EVENT_END_TASK` message when the user tries to shut down Windows or when the user tries to terminate your application, for example, when the user executes the **Close** command from the task bar button for your application under Windows 95/NT.

The tutorial in the *Getting Started with LabWindows/CVI* manual presents examples of callback functions. 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.

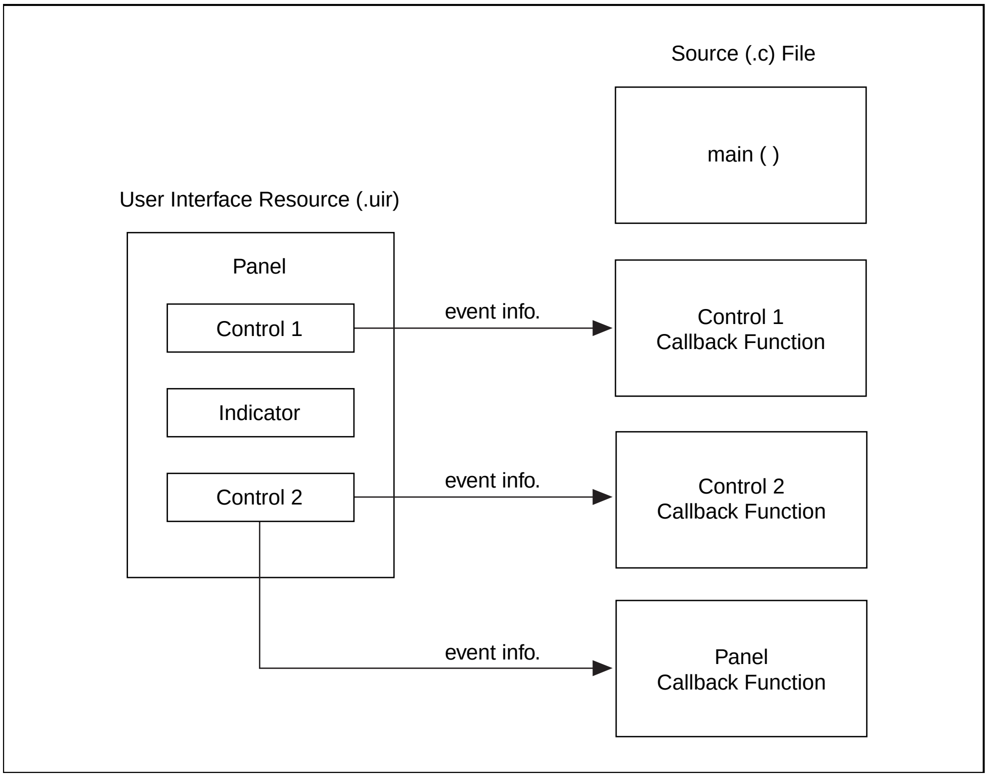


Figure 3-1. Callback Function Concept

```
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 CONTROL1                */
                  break;
               case EVENT_VAL_CHANGED :
                  ./* Code that responds to a value    */
                  ./* change on CONTROL1               */
                  break;
               case EVENT_COMMIT :
                  ./* Code that responds to a commit   */
                  ./* event on CONTROL1                */
                  break;
            }
         }
         if (control == PANEL_CONTROL2) {
            switch (event) {
               case EVENT_RIGHT_CLICK :
                  ./* Code that responds to a right    */
                  ./* click on CONTROL2                */
                  break;
               case EVENT_COMMIT :
                  ./* Code that responds to a commit   */
                  ./* event on 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    */
            .                   /* MENU1 of the menu bar.            */
            break;
        case MENUBAR_MENU1_ITEM2:
            .                   /* Code that responds to ITEM2 in    */
            .                   /* MENU1 of the menu bar.            */
            break;
    }
    return(0);
}
```

Note ***If you assign callback functions to your GUI objects using the User Interface Editor,*** `LoadPanel` ***and*** `LoadMenuBar` ***automatically install these functions. Otherwise, you must install them programmatically through the callback installation functions (***`InstallPanelCallback`***,*** `InstallCtrlCallback`***,*** `InstallMenuCallback`***, and others).***

Note ***Do not call*** `longjmp` ***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, refer to 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

`GetUserEvent` returns only commit events and events you post 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 number and pressing <Enter>. Figure 3-2 illustrates the event-loop concept of `GetUserEvent`.

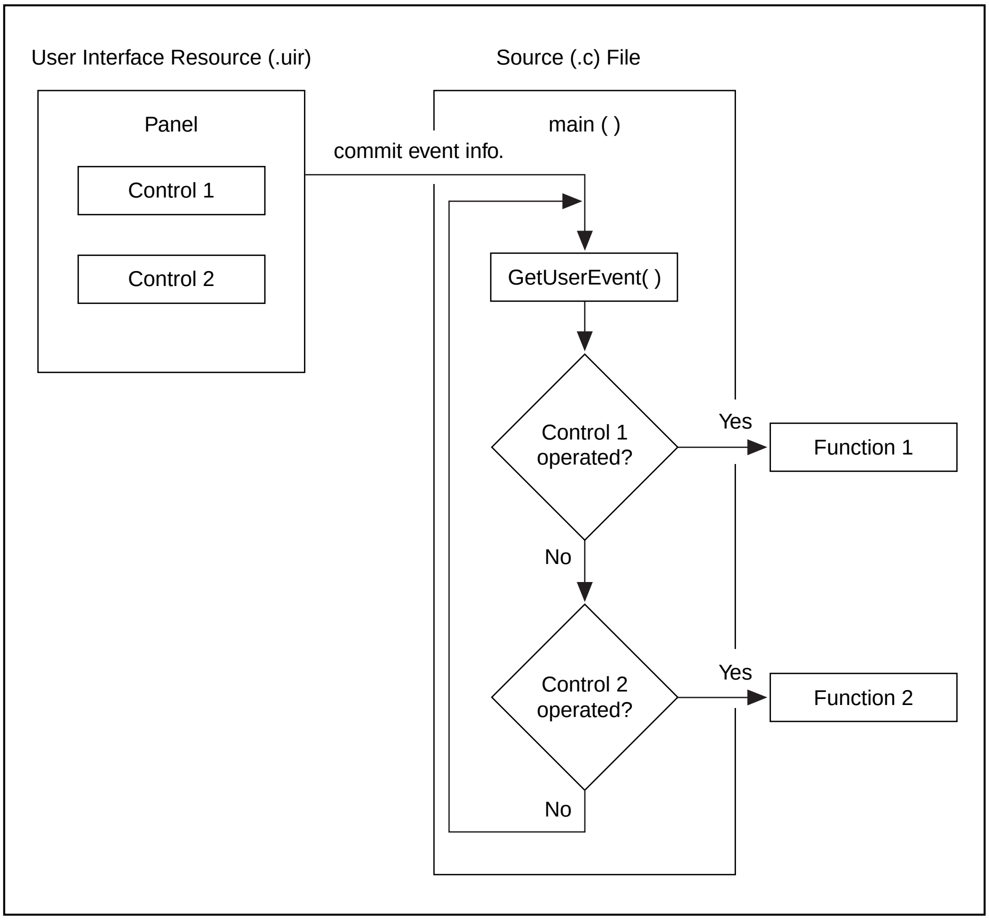


Figure 3-2. Event Loop Concept

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

```
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;
    }
}
```

If you use `GetUserEvent`, you can also install callbacks to receive events other than commit events.

You should read the remainder of this chapter 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 this chapter and Chapter 1, *User Interface Concepts*.

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

`LoadPanel` loads into memory a panel you created in the User Interface Editor and saved in a `.uir` file. When you use `LoadPanel` refer to 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 refer to 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 `LoadPanelEx`, instead of `LoadPanel`, to load a panel in a DLL if the DLL contains the callback functions for the panel and its controls.

You use the following functions to manipulate panels in a user interface:

`NewPanel` creates 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.

`DuplicatePanel` creates 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. 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.

`DisplayPanel` displays 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.

`HidePanel` hides 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 shows the new plots. A hidden panel cannot generate events.

`DefaultPanel` 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 `SetCtrlAttribute` using `ATTR_DFLT_VALUE`.

`GetActivePanel` obtains the handle of the currently active panel.

`SetActivePanel` makes a particular panel active when multiple panels are visible.

`GetPanelAttribute` obtains a particular attribute of a panel. The *Panel Attributes* section of this chapter lists panel attributes.

`SetPanelAttribute` sets a particular attribute of a panel. The *Panel Attributes* section of this chapter lists panel attributes.

`SetPanelPos` changes the position of a panel on the screen. If the panel is visible, it is redrawn in its new position.

`SetPanelSize` changes the size of a panel. If the panel is visible, it is redrawn with its new size.

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

`RecallPanelState` recalls 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.

`DiscardPanel` removes a panel from memory. If the panel is visible, it is removed from the screen.

Programming with Pop-Up Panels

You use the following functions to control pop-up panels in a user interface:

`InstallPopup` displays and activates a panel as a dialog box. You must load a panel with `LoadPanel` or create the panel using `NewPanel`.

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 process only commit events. `GetUserEvent` returns the ID of the control that caused the event. `GetUserEvent` can operate in one of two ways.

- Waits for the user to generate an event before returning to the calling program.
- Returns immediately whether or not an event has occurred.

`RemovePopup` removes either the active pop-up panel or all pop-up panels. RemovePopup does not unload the panel from memory.

`SetSystemPopupAttributes` and `GetSystemPopupAttributes` set and obtain the values of attributes that affect all of the pop-up panel. You use the following functions to access the predefined pop-up panels:

```
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.

Role of Child Panels

You can configure `LoadPanel` 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 might be partially or completely hidden in the shrunken panel view.

Processing Panel Events

If you want 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, refer to 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

Name	Type	Description
`ATTR_ACTIVE`	integer	Indicates whether the panel or one of its child panels is the active panel (`GetPanelAttribute` only).
`ATTR_ACTIVATE_WHEN_CLICKED_ON`	integer	`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`	integer	RGB value. Refer to discussion following this table.
`ATTR_CALLBACK_DATA`	void *	A pointer to data that you can define. The pointer 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 *	Name of the callback function associated with the panel (`GetPanelAttribute` only).
`ATTR_CALLBACK_NAME_LENGTH`	integer	Number of characters in the name of the panel callback (`GetPanelAttribute` only).
`ATTR_CAN_MAXIMIZE`	integer	Windows only. `1` = you can maximize the top-level panel `0` = you cannot maximize the top-level panel
`ATTR_CAN_MINIMIZE`	integer	Windows only. `1` = you can minimize the top-level panel `0` = you cannot minimize the top-level panel
`ATTR_CLOSE_CTRL`	integer	ID of control that receives commit events when a user selects the **Close** command in the system menu (top-level panels only).
`ATTR_CLOSE_ITEM_VISIBLE`	integer	Specifies whether the **Close** item is available in the system menu of the top-level panel. Windows only. `0` = Dimmed Close Item `1` = Enable Close Item

Table 3-2. Panel Attributes (Continued)

Name	Type	Description
`ATTR_CONFORM_TO_SYSTEM`	integer	Specifies whether the panel and its controls use the system colors. Subsequent new controls also use the system colors. This attribute is useful only on Windows 95/NT. On other platforms, LabWindows/CVI always uses panel gray and black as the system colors.
`ATTR_CONSTANT_NAME`	char *	Constant name you assigned to the panel in the User Interface Editor (`GetPanelAttribute` only).
`ATTR_CONSTANT_NAME_LENGTH`	integer	Number of characters in the constant name of the panel (`GetPanelAttribute` only).
`ATTR_DIMMED`	integer	`1` = panel disabled `0` = panel enabled
`ATTR_FIRST_CHILD`	integer	Panel handle of the first child panel in the panel you specify. (`GetPanelAttribute` only). First top-level panel if you specify a panel handle of zero. Zero if no child panels. To get the handle of the next child panel, use `ATTR_NEXT_PANEL` with the first child panel handle.
`ATTR_FLOATING`	integer	Specifies whether the panel floats above all non-floating panels. Applies to top-level panels in Windows.
`ATTR_FRAME_COLOR`	integer	RGB value (child panels only). Refer to discussion following this table.
`ATTR_FRAME_STYLE`	integer	Frame style of the panel (child panels only).
`ATTR_FRAME_THICKNESS`	integer	Panel frame thickness (child panels only). Range = 1 to 10. Refer to Figure 3-3.
`ATTR_HAS_TASKBAR_BUTTON`	integer	`1` = Panel has its own taskbar button. The button text is always the title of the panel. `0` = No separate taskbar button for panel. Applies only to top-level panels on Windows 95/NT.

Table 3-2. Panel Attributes (Continued)

Name	Type	Description
`ATTR_HEIGHT`	integer	Height of panel, excluding the frame and title bar. Range = 1 to 32,767. Refer to Figure 3-3.
`ATTR_HSCROLL_OFFSET`	integer	The offset of the horizontal scroll bar (in pixels).
`ATTR_HSCROLL_OFFSET_MAX`	integer	Number of pixels you have to scroll to reach the right edge of the right-most object on the panel (`GetPanelAttribute` only).
`ATTR_LEFT`	integer	Left position of panel, range = 1 to 32,767 or `VAL_AUTO_CENTER`. Refer to Figure 3-3.
`ATTR_MENU_BAR_VISIBLE`	integer	`1` = menu bar visible `0` = menu bar invisible
`ATTR_MENU_HEIGHT`	integer	Height of menu bar, range = 1 to 32,767 (`GetPanelAttribute` only). Refer to Figure 3-3.
`ATTR_MIN_HEIGHT_FOR_SCALING`	integer	Smallest panel height for which scaling is allowed. Range = 0 to 32,767
`ATTR_MIN_WIDTH_FOR_SCALING`	integer	Smallest panel width for which scaling is allowed. Range = 0 to 32,767
`ATTR_MOUSE_CURSOR`	integer	The mouse cursor style. Refer to Table 3-4.
`ATTR_MOVABLE`	integer	`1` = movable panel `0` = immovable panel (child panels only)
`ATTR_NEXT_PANEL`	integer	Panel handle of the next sibling panel. If no more siblings, `0`. Use `ATTR_FIRST_CHILD` to get the handle of the first sibling panel. (`GetPanelAttribute` only).
`ATTR_NUM_CHILDREN`	integer	Number of child panels in the panel you specify (`GetPanelAttribute` only).
`ATTR_NUM_CTRLS`	integer	The number of controls on the panel (`GetPanelAttribute` only).

Table 3-2. Panel Attributes (Continued)

Name	Type	Description
`ATTR_PANEL_FIRST_CTRL`	integer	ID of the first control on the panel. If panel has no controls, `0`. Use `ATTR_NEXT_CHILD` with `GetCtrlAttribute` to get ID of the next control. (`GetPanelAttribute` only).
`ATTR_PANEL_MENU_BAR_CONSTANT`	char *	Resource ID constant name of the menu bar attached to the panel (`GetPanelAttribute` only).
`ATTR_PANEL_MENU_BAR_CONSTANT_LENGTH`	integer	Number of characters in the resource ID constant name of the menu bar attached to the panel (`GetPanelAttribute` only).
`ATTR_PANEL_PARENT`	integer	Panel handle of the parent panel. Zero, if the panel you specify is top level. (`GetPanelAttribute` only.)
`ATTR_PARENT_SHARES_SHORTCUT_KEYS`	integer	`1` = share shortcut keys `0` = do not share shortcut keys
`ATTR_RESOLUTION_ADJUSTMENT`	integer	Specifies degree of scaling for the panel and its contents on screens with differing resolutions. Values are 0 to 100 (percentage). (`GetPanelAttribute` only.) To override the setting in the `.uir` file, call `SetSystemAttribute` on this attribute before you call `LoadPanel`.
`ATTR_SCALE_CONTENTS_ON_RESIZE`	integer	Specifies scaling for panel contents when you resize the panel programmatically or interactively. `1` = Scale contents on resize `0` = Do not scale contents on resize
`ATTR_SCROLL_BAR_COLOR`	integer	RGB value. Refer to discussion following this table.
`ATTR_SCROLL_BARS`	integer	`VAL_NO_SCROLL_BARS` or `VAL_HORIZ_SCROLL_BAR` or `VAL_VERT_SCROLL_BAR` or `VAL_BOTH_SCROLL_BARS`

Table 3-2. Panel Attributes (Continued)

Name	Type	Description
`ATTR_SIZABLE`	integer	`1` = panel sizable `0` = panel not sizable (child panels only).
`ATTR_SYSTEM_MENU_VISIBLE`	integer	`1` = The system menu is visible `0` = The system menu is not visible
`ATTR_SYSTEM_WINDOW_HANDLE`	integer	Returns a number you can cast to obtain the system specific window handle for a top-level window (`GetPanelAttribute` only). Refer to discussion following this table.
`ATTR_TITLE`	char *	Title of the panel.
`ATTR_TITLE_BACKCOLOR`	integer	RGB value (child panels only). Refer to discussion following this table.
`ATTR_TITLE_BOLD`	integer	`1` = title is bold `0` = title is not bold (child panels only).
`ATTR_TITLE_COLOR`	integer	RGB value (child panels only). Refer to discussion following this table.
`ATTR_TITLE_FONT`	char *	Font of the panel title (child panels only). Refer to Table 3-5.
`ATTR_TITLE_FONT_NAME_LENGTH`	integer	Number of characters in the name of the title font (`GetPanelAttribute` only; child panels only).
`ATTR_TITLE_ITALIC`	integer	`1` = title in italics `0` = title not in italics (child panels only).
`ATTR_TITLE_LENGTH`	integer	Number of characters in panel title (`GetPanelAttribute` only).
`ATTR_TITLE_POINT_SIZE`	integer	Point size of the title. Range = 1 to 32,767

Table 3-2. Panel Attributes (Continued)

Name	Type	Description
`ATTR_TITLE_SIZE_TO_FONT`	integer	Specify whether the child panel title is sized to the font. `0` = not sized `1` = sized
`ATTR_TITLE_STRIKEOUT`	integer	`1` = title has strikeout `0` = title does not have strikeout (child panels only).
`ATTR_TITLE_UNDERLINE`	integer	`1` = title underlined `0` = title not underlined (child panels only).
`ATTR_TITLEBAR_THICKNESS`	integer	Thickness of panel title bar, range = 1 to 32,767 (child panels only). Refer to Figure 3-3.
`ATTR_TITLEBAR_VISIBLE`	integer	`1` = title bar visible `0` = title bar invisible
`ATTR_TOP`	integer	Top position of panel, range = 1 to 32,767 or `VAL_AUTO_CENTER`. Refer to Figure 3-3.
`ATTR_VISIBLE`	integer	`1` = panel visible `0` = panel invisible
`ATTR_VSCROLL_OFFSET`	integer	Offset of the vertical scroll bar, in pixels.
`ATTR_VSCROLL_OFFSET_MAX`	integer	Number of pixels you have to scroll to reach the bottom edge of the lowest object on the panel (`GetPanelAttribute` only).
`ATTR_WIDTH`	integer	Width of the panel in pixels excluding the text frame. Valid range: 0 to 32,767
`ATTR_WINDOW_ZOOM`	integer	`VAL_MAXIMIZE`, `VAL_MINIMIZE`, or `VAL_NO_ZOOM` (Windows only). Refer to the *Panel Attribute Discussion*.
`ATTR_ZPLANE_POSITION`	integer	Drawing order for child panels. Use `0` to draw on top of other child panels. Valid Range: 0 to (Number of Panels – 1)

Panel Attribute Discussion

When you resize a parent panel for which ATTR_SCALE_CONTENTS_ON_RESIZE is enabled, scaling occurs for controls and child panels within the parent panel. To make scaling occur within child panels you must enable ATTR_SCALE_CONTENTS_ON_RESIZE for each child panel.

When you resize a panel for which ATTR_SCALE_CONTENTS_ON_RESIZE is enabled and the new size falls below minimum values you have specified for ATTR_MIN_HEIGHT_FOR_SCALING or ATTR_MIN_WIDTH_FOR_SCALING, scaling occurs to those minimum values and the panel clips part of the contents.

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

ATTR_SYSTEM_WINDOW_HANDLE returns an integer you can 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 under Windows, and Window under X Windows. Setting other panel attributes, for example, ATTR_TITLEBAR_VISIBLE, might cause the value of ATTR_SYSTEM_WINDOW_HANDLE to change because the panel has to destroy and recreate its window. Installing and uninstalling a panel as a pop-up can also change the window handle. Thus, obtain the window handle again after taking any of these actions.

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

Setting the ATTR_WINDOW_ZOOM attribute automatically makes the panel visible. If you set the attribute to VAL_NO_ZOOM or VAL_MAXIMIZE, the panel is activated. If you set the attribute to VAL_MINIMIZE, the panel is deactivated. Hiding the panel 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 Table 3-3. The first 16 colors listed are the 16 standard colors.

Table 3-3. Common Color Values

Value	Code
VAL_RED	0xFF0000L
VAL_GREEN	0x00FF00L
VAL_BLUE	0x0000FFL
VAL_CYAN	0x00FFFFL

Table 3-3. Common Color Values (Continued)

Value	Code
VAL_MAGENTA	0xFF00FFL
VAL_YELLOW	0xFFFF00L
VAL_DK_RED	0x800000L
VAL_DK_BLUE	0x000080L
VAL_DK_GREEN	0x008000L
VAL_DK_CYAN	0x008080L
VAL_DK_MAGENTA	0x800080L
VAL_DK_YELLOW	0x808000L
VAL_LT_GRAY	0xCCCCCCL
VAL_DK_GRAY	0x808080L
VAL_BLACK	0x000000L
VAL_WHITE	0xFFFFFFL
VAL_PANEL_GRAY	0xCCCCCCL
VAL_GRAY	0xA0A0A0L
VAL_OFFWHITE	0xE5E5E5L
VAL_TRANSPARENT	0x1000000L

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.

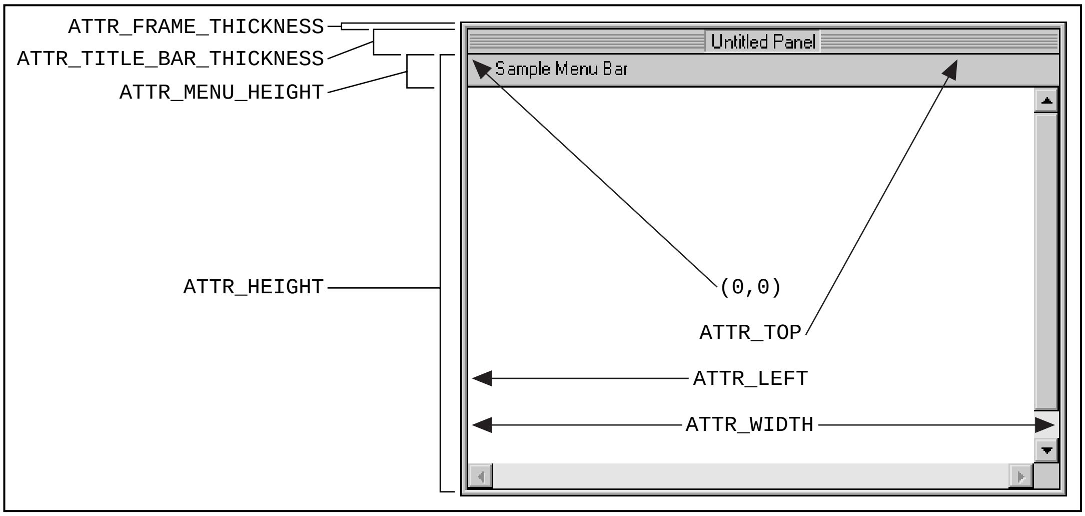


Figure 3-3. Geometric Attributes of a Panel

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

Value	Cursor Style
VAL_DEFAULT_CURSOR	↖
VAL_CHECK_CURSOR	✓
VAL_CROSS_HAIR_CURSOR	+
VAL_BOX_CURSOR	□
VAL_POINTING_FINGER_CURSOR	👆
VAL_OPEN_HAND_CURSOR	✋
VAL_QUESTION_MARK_CURSOR	?

Table 3-4. Values and Cursor Styles for ATTR_MOUSE_CURSOR (Continued)

Value	Cursor Style
`VAL_HOUR_GLASS_CURSOR`	⌛
`VAL_HIDDEN_CURSOR`	
`VAL_SIZE_NS_CURSOR`	⇕
`VAL_SIZE_EW_CURSOR`	⇔
`VAL_SIZE_NW_SE_CURSOR`	⤡
`VAL_SIZE_NE_SW_CURSOR`	⤢

Table 3-5 lists valid font values.

Table 3-5. Font Values

Type	Value
Platform-independent fonts	`VAL_MENU_FONT`
	`VAL_MESSAGE_BOX_FONT`
	`VAL_DIALOG_FONT`
	`VAL_EDITOR_FONT`
	`VAL_APP_FONT`
Platform-independent metafonts	`VAL_MENU_META_FONT`
	`VAL_MESSAGE_BOX_META_FONT`
	`VAL_DIALOG_META_FONT`
	`VAL_EDITOR_META_FONT`
	`VAL_APP_META_FONT`
LabWindows/CVI supplied metafonts	`VAL_7SEG_META_FONT` `VAL_SYSTEM_META_FONT`
Host fonts	Refer to discussion following this table.
User-defined metafonts	Refer to discussion following this table.

Platform-Independent Fonts That Are Resident on PCs and UNIX

The platform-independent fonts contain typeface information only and use typefaces native to each platform supported by LabWindows/CVI.

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. `VAL_EDITOR_META_FONT` is a monospaced metafont that is the default font in the Source windows. `VAL_APP_META_FONT` is a smaller, non-bold alternative to `VAL_DIALOG_META_FONT` and is the default font in the Project window. `VAL_MESSAGE_BOX_META_FONT` is used for simple message boxes.

Metafonts Supplied by LabWindows/CVI

National Instruments ships LabWindows/CVI with two metafonts that 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. Avoid using the extended IBM characters because other fonts do not display 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 `CreateMetaFont`. You can then apply the metafont to any control or panel attribute that takes a font value. When you apply a metafont, you set the typeface, point size, and text styles all at one time. 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

You use the following functions to control menu bars in a user interface:

`LoadMenuBar` loads into memory a menu bar you created in the User Interface Editor and and saved in a `.uir` file. When you call `LoadMenuBar`, refer to 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 refer to the menu bar. Use `LoadMenuBarEx`, instead of `LoadMenuBar`, to load a panel from a DLL if the DLL contains the callback functions for the menu callbacks. After you load a menu bar, it generates user events when the user selects menu bar commands. Refer to the *Processing Menu Bar Events* section of this chapter for additional information.

`NewMenuBar` creates 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.

`SetPanelMenuBar` assigns a menu bar to a panel. You can assign a panel only one menu bar at a time. Multiple panels can share the same menu bar.

`GetPanelMenuBar` gets the menu bar handle associated with a panel.

`GetSharedMenuBarEventPanel` gets the handle of the panel on which a menu bar event occurred. This is useful when multiple panels share a menu bar. You need this function only when using `GetUserEvent`; menu callback functions receive the panel handle as a parameter.

`GetMenuBarAttribute` obtains a particular menu bar or menu item attribute.

`SetMenuBarAttribute` sets a particular menu bar or menu item attribute.

`InsertSeparator` programmatically places a dividing line between menu items.

`NewMenu` / `DiscardMenu`, `NewMenuItem` / `DiscardMenuItem`, and `NewSubMenu` / `DiscardSubMenu` programmatically alters a menu system.

`EmptyMenu` removes the contents of a particular menu or submenu.

`EmptyMenuBar` removes the menu bar from the screen and destroys its contents, but does not destroy the menu bar handle.

`DiscardMenuBar` removes the menu bar from the screen and free its handle from memory.

Processing Menu Bar Events

Two ways exist for processing 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 use `NewMenuBar` to create your menu bar programmatically, 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. 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 shortcut key. In this callback function, you can update the state of menu items before the pull-down menu appears. You can add or delete menu items, dim or undim menu items, add or remove checkmarks next to menu items, or change the names of menu items. Use the `MenuDimmerCallbackPtr` typedef from `userint.h` as a model to declare this callback function in your program.

You initiate the callback mechanism by calling `RunUserInterface` or a `GetUserEvent` loop.

Using an Event Loop

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

- Waits for the user to generate an event before returning to the calling program.
- 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 3-6. Menu and Menu Item Attributes

Attribute	Name	Type	Description
Menu and Menu Item Attribute	`ATTR_DIMMED`	integer	`1` = menu/item is disabled `0` = menu/item is enabled Pass `0` (zero) for the menu ID if the attribute corresponds to the entire menu bar.
	`ATTR_CALLBACK_DATA`	void *	A pointer to data that you can define. The pointer is 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 *	Name of the callback function associated with the menu or menu item (`GetMenuBarAttribute` only).
	`ATTR_CALLBACK_NAME_LENGTH`	integer	Number of characters in the name of the menu or menu item callback name (`GetMenuBarAttribute` only).

Table 3-6. Menu and Menu Item Attributes (Continued)

Attribute	Name	Type	Description
Menu and Menu Item Attribute (continued)	`ATTR_CONSTANT_NAME`	char *	Constant name you assigned to the menu or menu item in the User Interface Editor (`GetMenuBarAttribute` only).
	`ATTR_CONSTANT_NAME_LENGTH`	integer	Number of characters in the constant name of the menu or menu item (`GetMenuBarAttribute` only).
Menu	`ATTR_MENU_NAME`	char *	The name of the menu item.
	`ATTR_MENU_NAME_LENGTH`	integer	The number of characters in the menu item (`GetMenuBarAttribute` only).
Menu Bar Only	`ATTR_DRAW_LIGHT_BEVEL`	integer	Indicates whether the menu bar draws with a light or dark bevel at the bottom; applies only to Windows 95. `0` = Dark Bevel (the default) `1` = Light Bevel
	`ATTR_NUM_MENUS`	integer	Number of menus for the menu bar.
Menu Item	`ATTR_CHECKED`	integer	`1` = a check mark by the menu item `0` = no check mark by the menu item
	`ATTR_IS_SEPARATOR`	integer	`1` = the menu item is a separator `0` = the menu item is not a separator (`GetMenuBarAttribute` only).

Table 3-6. Menu and Menu Item Attributes (Continued)

Attribute	Name	Type	Description
Menu Item (continued)	ATTR_ITEM_NAME	char *	The name of the menu item
	ATTR_ITEM_NAME_LENGTH	integer	The number of characters in the menu item name (`GetMenuBarAttribute` only).
	ATTR_NUM_MENU_ITEMS	integer	Number of menu items for the menu.
	ATTR_SHORTCUT_KEY	integer	Key code for the menu item shortcut key. Refer to Table 3-7.
	ATTR_SUBMENU_ID	integer	Resource ID for the submenu (`GetMenuBarAttribute` only).

Menu Bar Attribute Discussion

In source code, a shortcut key is represented by a 4-byte integer consisting of three bit fields, `0x00MMVVAA`, where:

`MM` = the modifier key

`VV` = the virtual key

`AA` = the ASCII key

When you construct a shortcut key, the modifier key is bitwise OR'ed with a virtual key or an ASCII key. Either the ASCII field or the virtual key field must be zero. 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

Type	Value
Key modifiers	`VAL_SHIFT_MODIFIER`
	`VAL_SHIFT_AND_MENUKEY`
	`VAL_MENUKEY_MODIFIER` <Ctrl> on the PC, and <Ctrl> or <Meta> on the SPARCstation.
	`VAL_UNDERLINE_MODIFIER` <Alt> on the PC, and <Alt> or <Meta> on the SPARCstation.
Virtual key codes	`VAL_FWD_DELETE_VKEY` not available on SPARCstation keyboards.
	`VAL_BACKSPACE_VKEY` <del> and <Back Space> on the SPARCstation.
	`VAL_ESC_VKEY`
	`VAL_TAB_VKEY`
	`VAL_ENTER_VKEY`
	`VAL_UP_ARROW_VKEY`
	`VAL_DOWN_ARROW_VKEY`
	`VAL_LEFT_ARROW_VKEY`
	`VAL_RIGHT_ARROW_VKEY`
	`VAL_INSERT_VKEY`
	`VAL_HOME_VKEY`
Virtual key codes	`VAL_END_VKEY`
	`VAL_PAGE_UP_VKEY`
	`VAL_PAGE_DOWN_VKEY`
	`VAL_F1_VKEY`
	`VAL_F2_VKEY`
	`VAL_F3_VKEY`
	`VAL_F4_VKEY`

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

Type	Value
Virtual key codes (continued)	`VAL_F5_VKEY`
	`VAL_F6_VKEY`
	`VAL_F7_VKEY`
	`VAL_F8_VKEY`
	`VAL_F9_VKEY`
	`VAL_F10_VKEY`
Virtual key codes	`VAL_F11_VKEY`
	`VAL_F12_VKEY`

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

Value	Mask
`VAL_MODIFIER_MASK`	`0x00FF0000`
`VAL_VKEY_MASK`	`0x0000FF00`
`VAL_ASCII_KEY_MASK`	`0x000000FF`

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 `.uir` file into memory, it also loads the controls on that panel. To refer to a particular control in a call to a User Interface Library function, use the defined constant you assigned to the control in the User Interface Editor. The defined constant evaluates to the resource ID for the control. You use the following functions to manage general aspects of all controls:

`NewCtrl` creates 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`.

`DuplicateCtrl` creates a new control that is a duplicate of a control that you loaded with `LoadPanel` or created with `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`.

`SetCtrlVal` sets a control to a particular value. *This function is not valid for Graph and Strip Chart controls.*

`GetCtrlVal` obtains the current value of a control. *This function is not valid for Graph and Strip Chart controls.*

`GetActiveCtrl` obtains the control that receives keyboard events when its panel is the active panel.

`SetActiveCtrl` establishes 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.

`DefaultCtrl` restores 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.*

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

`GetCtrlAttribute` obtains a particular attribute of a control.

`SetCtrlAttribute` sets a particular attribute of a control. When changing control attributes that affect the font of a control, set the `ATTR_TEXT_FONT` attribute first.

`DiscardCtrl` removes a control from memory. If its panel is visible, the control is removed from the screen. LabWindows/CVI does not allow you to call `DiscardCtrl` from the callback function for the control, except in response to a commit event. A call to `DiscardCtrl` from any other type of event might 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, refer to the *Programming with Picture Controls* section in this chapter.

List Boxes and Rings

You use the following functions to manage list controls:

`InsertListItem` adds an item to the list.

`DeleteListItem` deletes one or more items from the list.

`ReplaceListItem` replaces a given list item with a new item. To preserve the existing label and change only the value, pass `0` (zero) as the label parameter.

`GetCtrlVal` obtains the value of the currently active list item.

`SetCtrlVal` sets the currently active list item to the first item in the list that has a given value.

`GetCtrlIndex` obtains the zero-based index of the currently active list item.

`SetCtrlIndex` sets the currently active list item based on a given zero-based index.

`GetValueFromIndex` obtains the value of the list item corresponding to a given zero-based index into the list.

`GetValueLengthFromIndex` obtains the length of the value (strings only) of the list item corresponding to a given zero-based index into the list.

`GetIndexFromValue` obtains a zero-based index of the first list item that has a given value.

`GetNumListItems` obtains the number of items contained in a list control.

`GetLabelFromIndex` obtains the label of the list item corresponding to a zero-based index.

`GetLabelLengthFromIndex` obtains the length of the label of the list item corresponding to a zero-based index.

`ClearListCtrl` clears the contents of a list control.

List Boxes Only

You use the following functions to manage list boxes only:

`SetListItemImage` sets the image associated with a particular list item. `GetListItemImage` obtains the image associated with a particular list item. Examples of list item images are the folder icons that appear in the Open/Save File dialog box in LabWindows/CVI.

`IsListItemChecked` determines if a particular list item is checked.

`CheckListItem` programmatically checks a list item.

`GetNumCheckedItems` determines 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 32,767, the text box does not scroll. If you use `NIDialogMetaFont` or the `NIEditorMetaFont`, your text box can contain a maximum of approximately 2,500 lines. You use the following functions to manage text boxes:

`GetCtrlVal` obtains the text in the text box.

`SetCtrlVal` appends text to a text box.

`InsertTextBoxLine` adds text into a text box starting at a given line.

`DeleteTextBoxLine` deletes one or more lines from a text box.

`ReplaceTextBoxLine` replaces a given line with new text.

`GetNumTextBoxLines` obtains the number of lines used in a text box.

`GetTextBoxLineLength` obtains the length of a particular line in a text box.

`GetTextBoxLineOffset` obtains the byte offset of the beginning of a line in a text box from the beginning of the entire text.

`GetTextBoxLine` obtains the text of a particular line in a text box.

`GetTextBoxLineIndexFromOffset` obtains the zero-based index of the line that contains the character at a given byte offset beyond the beginning of the entire text.

`ResetTextBox` replaces the contents of a text box.

Processing Control Events

When a panel is displayed, two ways exist for processing 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 use `NewCtrl` to create your controls programmatically, 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, except in response to a commit event. Doing this gives unpredictable results.***

Using an Event Loop

When the user generates a commit event on a control, `GetUserEvent` returns the appropriate ID of the control as a parameter. `GetUserEvent` can operate in one of two ways.

- Waits for the user to generate an event before returning to the calling program.
- Returns immediately whether or not an event occurs. If no event occurs, the control ID parameter is `-1`.

Control Attributes

Tables 3-9 to 3-44 list control attributes that you can retrieve or change through `GetCtrlAttribute` and `SetCtrlAttribute`.

Table 3-9. Control Attributes for All Controls

Name	Type	Description
`ATTR_CALLBACK_DATA`	void *	A pointer to data that you can define. The pointer is passed to the panel callback function.
`ATTR_CALLBACK_FUNCTION_POINTER`	void *	A pointer to the callback function for the control.
`ATTR_CALLBACK_NAME`	char *	Name of the callback function associated with the control (`GetCtrlAttribute` only).
`ATTR_CALLBACK_NAME_LENGTH`	integer	Number of characters in the name of the control callback name (`GetCtrlAttribute` only).
`ATTR_CONSTANT_NAME`	char *	Constant name you assigned to the control in the User Interface Editor (`GetCtrlAttribute` only).
`ATTR_CONSTANT_NAME_LENGTH`	integer	Number of characters in the constant name of the control (`GetCtrlAttribute` only).
`ATTR_CTRL_STYLE`	integer	Control style (`GetCtrlAttribute` only). Refer to Table 3-45.
`ATTR_DIMMED`	integer	`1` = control disabled `0` = control enabled
`ATTR_LEFT`	integer	–32,768 to 32,767
`ATTR_NEXT_CTRL`	integer	ID of the next control on the panel. If no more controls, `0`. Use `ATTR_PANEL_FIRST_CTRL` with `GetPanelAttribute` to get the ID of the first control. (`GetCtrlAttribute` only.)
`ATTR_OVERLAPPED`	integer	Indicates whether the control is overlapped by another control or by one of its own parts (`GetCtrlAttribute` only).
`ATTR_TOP`	integer	–32,768 to 32,767

Table 3-9. Control Attributes for All Controls (Continued)

Name	Type	Description
`ATTR_VISIBLE`	integer	`1` = control visible `0` = control invisible
`ATTR_WIDTH`	integer	0 to 32,767
`ATTR_ZPLANE_POSITION`	integer	Order of the control in the z-plane. Lowest ordered control (`0`) is on top.

Table 3-10. Control Attributes for All Controls Except Simple Strings, Simple Numerics, and Simple Rings

Name	Type	Description
`ATTR_HEIGHT`	integer	0 to 32,767

Table 3-11. Control Attributes for All Controls Except Indicator-Only Controls

Name	Type	Description
`ATTR_CTRL_MODE`	integer	`VAL_HOT` or `VAL_NORMAL` or `VAL_VALIDATE` or `VAL_INDICATOR`
`ATTR_CTRL_TAB_POSITION`	integer	Tab order of the control.
Indicator-only controls are: decorations, strip charts, text messages, timers, and pictures.		

Table 3-12. Control Attributes for All Controls Except Decorations, Canvases, Graphs, and Strip Charts

Name	Type	Description
`ATTR_CTRL_VAL`	Same as control type	Same functionality as `GetCtrlVal` and `SetCtrlVal`.

Table 3-13. Control Attributes for All Controls with Frames

Name	Type	Description
`ATTR_FRAME_COLOR`	integer	RGB value. Refer to discussion following this section.
Includes text boxes, strings, numerics, simple rings, ring meters, ring gauges, list boxes, decorations, and pictures.		

Table 3-14. Control Attributes for All Controls with Labels

Name	Type	Description
`ATTR_LABEL_BOLD`	integer	`1` = label is bold `0` = label is not bold
`ATTR_LABEL_COLOR`	integer	RGB value. Refer to discussion following this section.
`ATTR_LABEL_FONT`	char *	The font of the control label. Refer to Table 3-5.
`ATTR_LABEL_FONT_NAME_LENGTH`	integer	Number of characters in the name of the label font (`GetCtrlAttribute` only).
`ATTR_LABEL_ITALIC`	integer	`1` = label in italics `0` = label not in italics
`ATTR_LABEL_POINT_SIZE`	integer	Point size of the label; range = 1 to 32,767
`ATTR_LABEL_STRIKEOUT`	integer	`1` = label has strikeout `0` = label does not have strikeout
`ATTR_LABEL_TEXT`	char *	The label of the control.
`ATTR_LABEL_TEXT_LENGTH`	integer	Number of characters in the label (`GetCtrlAttribute` only).
`ATTR_LABEL_UNDERLINE`	integer	`1` = label underlined `0` = label not underlined
`ATTR_LABEL_VISIBLE`	integer	`1` = label is visible `0` = label is invisible
All controls except decorations and text messages.		

Table 3-15. Control Attributes for Controls with Labels, Except Non-Picture Command Buttons

Name	Type	Description
`ATTR_LABEL_BGCOLOR`	integer	Label background color, RGB value.
`ATTR_LABEL_HEIGHT`	integer	Height of label; range = 0 to 32,767
`ATTR_LABEL_JUSTIFY`	integer	`VAL_LEFT_JUSTIFIED` or `VAL_RIGHT_JUSTIFIED` or `VAL_CENTER_JUSTIFIED`

Table 3-15. Control Attributes for Controls with Labels, Except Non-Picture Command Buttons (Continued)

Name	Type	Description
`ATTR_LABEL_LEFT`	integer	Left position of label, range = –32,768 to 32,767 or use `VAL_AUTO_CENTER`.
`ATTR_LABEL_RAISED`	integer	`1` = label is raised `0` = label is not raised
`ATTR_LABEL_SIZE_TO_TEXT`	integer	`1` = label border restricted to label text size `0` = label border not restricted to label text size
`ATTR_LABEL_TOP`	integer	Top position of label, range = –32,768 to 32,767 or use `VAL_AUTO_CENTER` or `VAL_RIGHT_ANCHOR` or `VAL_LEFT_ANCHOR`.
`ATTR_LABEL_WIDTH`	integer	Width of label, range = 0 to 32,767.

Table 3-16. Control Attributes for Controls with Text, Except Graphs and Strip Charts

Name	Type	Description
`ATTR_TEXT_BOLD`	integer	`1` = text is bold `0` = text is not bold
`ATTR_TEXT_COLOR`	integer	RGB value. Refer to discussion below.
`ATTR_TEXT_FONT`	char *	The font of the text. Refer to Table 3-5.
`ATTR_TEXT_FONT_NAME_LENGTH`	integer	Number of characters in the name of the text font (`GetCtrlAttribute` only).
`ATTR_TEXT_ITALIC`	integer	`1` = text in italics `0` = text not in italics
`ATTR_TEXT_POINT_SIZE`	integer	Point size of the text, range = 1 to 32,767.
Controls with text are all controls except decorations, pictures, canvases, LEDs, and buttons without on/off text.		

Table 3-16. Control Attributes for Controls with Text, Except Graphs and Strip Charts (Continued)

Name	Type	Description
`ATTR_TEXT_STRIKEOUT`	integer	`1` = text has strikeout `0` = text does not have strikeout
`ATTR_TEXT_UNDERLINE`	integer	`1` = text underlined `0` = text not underlined
Controls with text are all controls except decorations, pictures, canvases, LEDs, and buttons without on/off text.		

Table 3-17. Control Attributes for Controls with Text Except Graphs, Strip Charts, Ring Slides, Binary Switches, and Text Buttons

Name	Type	Description
`ATTR_TEXT_BGCOLOR`	integer	RGB value. Refer to discussion below.

Table 3-18. Control Attributes for Controls with Text, Except Graphs, Strip Charts, Ring Slides, Pop-Up Rings, Binary Switches, and Text Buttons

Name	Type	Description
`ATTR_TEXT_JUSTIFY`	integer	`VAL_LEFT_JUSTIFIED` or `VAL_RIGHT_JUSTIFIED` or `VAL_CENTER_JUSTIFIED`

Table 3-19. Control Attributes for Controls with Variable Data Types

Name	Type	Description
`ATTR_DATA_TYPE`	integer	Refer to Table 3-46.
Including numerics, rings, binary switches, and list boxes.		

Table 3-20. Control Attributes for Label/Value Controls (Slides, Rings, Binary Switches, and List Boxes)

Name	Type	Description
`ATTR_CTRL_INDEX`	integer	0 to 32,767
`ATTR_DFLT_INDEX`	integer	0 to 32,767

Table 3-21. Control Attributes for Numerics

Name	Type	Description
`ATTR_CHECK_RANGE`	integer	`VAL_COERCE` or `VAL_IGNORE` or `VAL_NOTIFY`. Refer to Table 3-48.
`ATTR_DFLT_VALUE`	Same as control type	Default value of the control.
`ATTR_FORMAT`	integer	Refer to Table 3-47.
`ATTR_INCR_VALUE`	Same as control type	Increments for INC/DEC arrows of the control.
`ATTR_MAX_VALUE`	Same as control type	Maximum value of the control.
`ATTR_MIN_VALUE`	Same as control type	Minimum value of the control.
`ATTR_PRECISION`	integer	The numeric precision, 0 to 15.
`ATTR_NSCROLL_OFFSET_ MAX`	integer	Maximum horizontal scroll bar offset in pixels; range 0 to 32,767.
`ATTR_SHOW_RADIX`	integer	`1` = radix is shown `0` = radix is not shown

Table 3-22. Control Attributes for Picture and Slide Rings and Numerics

Name	Type	Description
`ATTR_SHOW_INCDEC_ARROWS`	integer	`1` = INC/DEC arrows appear `0` = INC/DEC arrows are not shown

Table 3-23. Control Attributes for Strings and Text Boxes

Name	Type	Description
`ATTR_MAX_ENTRY_LENGTH`	integer	Maximum # of characters; –1 means no limit.
`ATTR_TEXT_SELECTION_LENGTH`	integer	0 to 32,767
`ATTR_TEXT_SELECTION_START`	integer	0 to 32,767

Table 3-24. Control Attributes for Text Messages, Strings, and Text Boxes

Name	Type	Description
`ATTR_STRING_TEXT_LENGTH`	integer	0 to 32,767 (`GetCtrlAttribute` only).

Table 3-25. Control Attributes for Text Boxes

Name	Type	Description
`ATTR_ENTER_IS_NEWLINE`	integer	Specifies whether <Enter> key causes a newline. `1` = <Enter> for newline `2` = <Ctrl-Enter> for newline
`ATTR_EXTRA_LINES`	integer	Number of lines retained off-screen (0 to 32,767). `-1` = unlimited number of lines retained off screen
`ATTR_HSCROLL_OFFSET`	integer	Horizontal scroll offset; range = 0 to 32,767 pixels.
`ATTR_TOTAL_LINES`	integer	Number of lines in text, both on screen and off (`GetCtrlAttribute` only).
`ATTR_WRAP_MODE`	integer	`VAL_CHAR_WRAP` or `VAL_LINE_WRAP` or `VAL_WORD_WRAP`

Table 3-26. Control Attributes for Text Boxes and List Boxes

Name	Type	Description
`ATTR_FIRST_VISIBLE_LINE`	integer	Index of first visible line; zero-based.
`ATTR_SCROLL_BAR_COLOR`	integer	Any valid RGB value.
`ATTR_SCROLL_BAR_SIZE`	integer	`VAL_SMALL_SCROLL_BARS` or `VAL_LARGE_SCROLL_BARS`
`ATTR_SCROLL_BARS`	integer	`VAL_NO_SCROLL_BARS` or `VAL_HORIZ_SCROLL_BAR` or `VAL_VERT_SCROLL_BAR` or `VAL_BOTH_SCROLL_BARS` Horizontal scroll bars are not valid for list boxes.
`ATTR_VISIBLE_LINES`	integer	Number of visible lines; 0 to 32,767.
`ATTR_TEXT_CLICK_TOGGLES_CHECK`	integer	`1` = clicking on the text area toggle the check mark `0` = clicking on the text area does not toggle the check mark

Table 3-27. Control Attributes for List Boxes

Name	Type	Description
`ATTR_ALLOW_ROOM_FOR_IMAGES`	integer	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 an image is currently 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).
`ATTR_CHECK_MODE`	integer	`1` = enable checking `0` = disable checking
`ATTR_CHECK_STYLE`	integer	`VAL_CHECK_MARK` or `VAL_CHECK_BOX`

Table 3-27. Control Attributes for List Boxes (Continued)

Name	Type	Description
`ATTR_HILITE_CURRENT_ITEM`	integer	Specifies whether to highlight the currently selected item in a list box. The highlight appears in reversed colors when list box is active, a dashed box when inactive.
`ATTR_TEXT_CLICK_TOGGLES_CHECK`	integer	`1` = clicking on the text area toggles the check mark `0` = clicking on the text area does not toggle the check mark

Table 3-28. Control Attributes for Strings, Numerics, and Text Boxes

Name	Type	Description
`ATTR_NO_EDIT_TEXT`	integer	`1` = enable text editing `0` = disable text editing

Table 3-29. Control Attributes for Text Messages

Name	Type	Description
`ATTR_TEXT_RAISED`	integer	`1` = text is raised `0` = text is not raised
`ATTR_SIZE_TO_TEXT`	integer	`1` = message border restricted to text size `0` = message border not restricted to text size

Table 3-30. Control Attributes for Command Buttons

Name	Type	Description
`ATTR_AUTO_SIZING`	integer	Set the command button to automatically resize when you programmatically change its text. Values: `VAL_ALWAYS_AUTO_SIZE`, `VAL_GROW_ONLY` (the default), or `VAL_NEVER_AUTO_SIZE.`
`ATTR_CMD_BUTTON_COLOR`	integer	RGB value. Refer to discussion that follows this table.
`ATTR_SHORTCUT_KEY`	integer	Refer to discussion of shortcut keys in *Menu Bar Attributes* section of this chapter.

Table 3-31. Control Attributes for Binary Switches

Name	Type	Description
`ATTR_BINARY_SWITCH_COLOR`	integer	RGB value. Refer to discussion that follows this section.
`ATTR_OFF_VALUE`	Same as control type	Value of control when OFF.
`ATTR_OFF_VALUE_LENGTH`	integer	Only if string value (`GetCtrlAttribute` only).
`ATTR_ON_VALUE`	Same as control type	Value of control when ON.
`ATTR_ON_VALUE_LENGTH`	integer	Only if string value (`GetCtrlAttribute` only).

Table 3-32. Control Attributes for LEDs and Buttons, Except Command Buttons

Name	Type	Description
`ATTR_OFF_COLOR`	integer	RGB value. Refer to discussion that follows this section.
`ATTR_ON_COLOR`	integer	RGB value. Refer to discussion that follows this section.

Table 3-33. Control Attributes for Text Buttons and Binary Switches

Name	Type	Description
`ATTR_OFF_TEXT`	char *	Text displayed in OFF position.
`ATTR_OFF_TEXT_LENGTH`	integer	Number of characters in the OFF text (`GetCtrlAttribute` only).
`ATTR_ON_TEXT`	char *	Text displayed in ON position.
`ATTR_ON_TEXT_LENGTH`	integer	Number of characters in the ON text (`GetCtrlAttribute` only).

Table 3-34. Control Attributes for Numerics with Digital Displays (Slides, Knobs, Dials, Meters, and Gauges)

Name	Type	Description
`ATTR_DIG_DISP_LEFT`	integer	Left position of the digital display; range = –32,768 to 32,767.
`ATTR_DIG_DISP_TOP`	integer	Top position of the digital display; range = –32,768 to 32,767.
`ATTR_DIG_DISP_HEIGHT`	integer	Height of the digital display, range 0 to 32,767.
`ATTR_DIG_DISP_WIDTH`	integer	Width of the digital display; range = 0 to 32,767.
`ATTR_SHOW_DIG_DISP`	integer	`1` = show digital display `0` = hide digital display

Table 3-35. Control Attributes for Numerics and Ring Slides, Knobs, Dials, Meters, and Gauges

Name	Type	Description
`ATTR_FILL_COLOR`	integer	RGB value. Refer to discussion that follows this section.
`ATTR_MARKER_STYLE`	integer	`VAL_NO_MARKERS` or `VAL_NO_INNER_MARKERS` or `VAL_FULL_MARKERS`
`ATTR_NEEDLE_COLOR`	integer	RGB value. Refer to discussion that follows this section.
`ATTR_SLIDER_COLOR`	integer	RGB value. Refer to discussion that follows this section.
`ATTR_TICK_STYLE`	integer	`VAL_NO_TICKS` or `VAL_NO_MINOR_TICKS` or `VAL_FULL_TICKS`

Table 3-36. Control Attributes for Numeric and Ring Slides

Name	Type	Description
`ATTR_FILL_HOUSING_COLOR`	integer	RGB value. Refer to discussion that follows this section.
`ATTR_FILL_OPTION`	integer	`VAL_NO_FILL`, `VAL_TOP_FILL`, `VAL_BOTTOM_FILL`, `VAL_RIGHT_FILL`, or `VAL_LEFT_FILL`
`ATTR_SLIDER_HEIGHT`	integer	0 to 32,767
`ATTR_SLIDER_WIDTH`	integer	0 to 32,767

Table 3-37. Control Attributes for Numeric and Ring Knobs, Dials, and Gauges

Name	Type	Description
`ATTR_MARKER_END_ANGLE`	integer	Range = 0 to 359
`ATTR_MARKER_START_ANGLE`	integer	Range = 0 to 359

Table 3-38. Control Attributes for Color Numerics

Name	Type	Description
`ATTR_SHOW_MORE_BUTTON`	integer	`1`/`0` = show/do not show the **More** button on the color numeric pop-up.
`ATTR_SHOW_TRANSPARENT`	integer	`1`/`0` = show/do not show the transparent icon on the color numeric pop-up.

Table 3-39. Control Attributes for Menu Rings

Name	Type	Description
`ATTR_MENU_ARROW_COLOR`	integer	RGB value. Refer to discussion that follows this section.

Table 3-40. Control Attributes for Timer Controls

Name	Type	Description
`ATTR_INTERVAL`	double	Interval at which the timer control callback function is called, in seconds.
`ATTR_ENABLED`	integer	`0` = timer callback is disabled `1` = timer callback is enabled

Table 3-41. Control Attributes for Picture Controls, Rings, and Buttons

Name	Type	Description
`ATTR_FIT_MODE`	integer	`VAL_SIZE_TO_IMAGE` or `VAL_SIZE_TO_PICTURE` or `VAL_PICT_CORNER` or `VAL_PICT_CENTER` or `VAL_PICT_TILE`

Table 3-42. Control Attributes for Picture Controls and Rings

Name	Type	Description
`ATTR_FRAME_VISIBLE`	integer	`1` = show picture frame `0` = hide picture frame

Table 3-43. Control Attributes for Picture Controls, Rings, and Canvas Controls

Name	Type	Description
`ATTR_PICT_BGCOLOR`	integer	RGB value. Refer to discussion that follows this section. Also applies to canvas controls

Table 3-44. Control Attributes for Picture Buttons (Command and Toggle)

Name	Type	Description
`ATTR_IMAGE_FILE`	char *	Filename of the image file to load into control. A `NULL` results in no image.
`ATTR_IMAGE_FILE_LENGTH`	integer	Length of the image filename.
`ATTR_SUBIMAGE_HEIGHT`	integer	Pixel height of subimage. Refer to discussion following this section.
`ATTR_SUBIMAGE_TOP`	integer	Top pixel coordinate of subimage. Refer to discussion following this section.
`ATTR_SUBIMAGE_LEFT`	integer	Left pixel coordinate of subimage. Refer to discussion following this section.
`ATTR_SUBIMAGE_WIDTH`	integer	Pixel width of subimage. Refer to discussion following this section.
`ATTR_USE_SUBIMAGE`	integer	`1` = enable subimage display `0` = disable subimage display Refer to discussion following this section.

Picture Button Subimage Discussion

`ATTR_USE_SUBIMAGE` allows the picture button to display a subset of its loaded image. You specify the subset of the image with 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 top left corner of the subimage, relative to the top left corner of the entire bitmap.

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

`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. Refer to Table 3-3 for a list of common color values.

Table 3-45 contains the control types that can be used with the `ATTR_CTRL_STYLE` attribute.

Table 3-45. Control Styles for ATTR_CTRL_STYLE

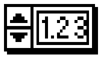
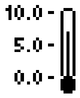
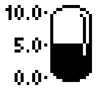
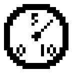
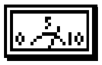
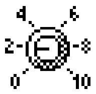
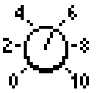
Type	Value	Icon
Numerics	`CTRL_NUMERIC`	
	`CTRL_NUMERIC_THERMOMETER`	
	`CTRL_NUMERIC_TANK`	
	`CTRL_NUMERIC_GAUGE`	
	`CTRL_NUMERIC_METER`	
	`CTRL_NUMERIC_KNOB`	
	`CTRL_NUMERIC_DIAL`	

Table 3-45. Control Styles for ATTR_CTRL_STYLE (Continued)

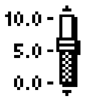
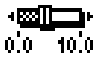
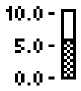
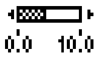
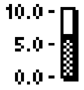
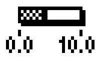
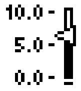
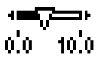
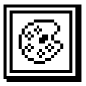
Type	Value	Icon
Numerics (continued)	`CTRL_NUMERIC_VSLIDE`	
	`CTRL_NUMERIC_HSLIDE`	
	`CTRL_NUMERIC_FLAT_VSLIDE`	
	`CTRL_NUMERIC_FLAT_HSLIDE`	
	`CTRL_NUMERIC_LEVEL_VSLIDE`	
	`CTRL_NUMERIC_LEVEL_HSLIDE`	
	`CTRL_NUMERIC_POINTER_VSLIDE`	
	`CTRL_NUMERIC_POINTER_HSLIDE`	
	`CTRL_COLOR_NUMERIC`	
String	`CTRL_STRING`	
Text message	`CTRL_TEXT_MSG`	
Text box	`CTRL_TEXT_BOX`	

Table 3-45. Control Styles for ATTR_CTRL_STYLE (Continued)

Type	Value	Icon
Command buttons	`CTRL_SQUARE_COMMAND_BUTTON`	OK
	`CTRL_OBLONG_COMMAND_BUTTON`	OK
	`CTRL_ROUND_COMMAND_BUTTON`	OK
	`CTRL_PICTURE_COMMAND_BUTTON`	🖼
	`CTRL_ROUNDED_COMMAND_BUTTON`	OK
Buttons	`CTRL_ROUND_BUTTON`	●
	`CTRL_SQUARE_BUTTON`	■
	`CTRL_ROUND_FLAT_BUTTON`	○
	`CTRL_SQUARE_FLAT_BUTTON`	□
	`CTRL_ROUND_RADIO_BUTTON`	◉
	`CTRL_SQUARE_RADIO_BUTTON`	▣
	`CTRL_CHECK_BOX`	☒
	`CTRL_ROUND_PUSH_BUTTON`	●
	`CTRL_SQUARE_PUSH_BUTTON`	■
	`CTRL_ROUND_PUSH_BUTTON2`	●

Table 3-45. Control Styles for ATTR_CTRL_STYLE (Continued)

Type	Value	Icon
Buttons (continued)	`CTRL_SQUARE_PUSH_BUTTON2`	▦
	`CTRL_SQUARE_TEXT_BUTTON`	ON
	`CTRL_OBLONG_TEXT_BUTTON`	ON
	`CTRL_ROUND_TEXT_BUTTON`	ON
	`CTRL_ROUNDED_TEXT_BUTTON`	ON
	`CTRL_PICTURE_TOGGLE_BUTTON`	▣
LEDs	`CTRL_ROUND_LIGHT`	●
	`CTRL_SQUARE_LIGHT`	■
	`CTRL_ROUND_LED`	●
	`CTRL_SQUARE_LED`	■
Binary switches	`CTRL_HSWITCH`	▨
	`CTRL_VSWITCH`	▨
	`CTRL_GROOVED_HSWITCH`	▨

Table 3-45. Control Styles for ATTR_CTRL_STYLE (Continued)

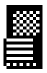
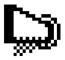
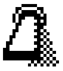
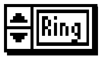
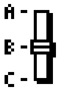
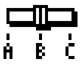
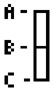
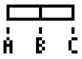
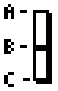
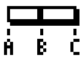
Type	Value	Icon
Binary switches	`CTRL_GROOVED_VSWITCH`	
	`CTRL_TOGGLE_HSWITCH`	
	`CTRL_TOGGLE_VSWITCH`	
Rings	`CTRL_RING`	
	`CTRL_RECESSED_MENU_RING`	
	`CTRL_MENU_RING`	
	`CTRL_POPUP_MENU_RING`	
	`CTRL_RING_VSLIDE`	
	`CTRL_RING_HSLIDE`	
	`CTRL_RING_FLAT_VSLIDE`	
	`CTRL_RING_FLAT_HSLIDE`	
	`CTRL_RING_LEVEL_VSLIDE`	
	`CTRL_RING_LEVEL_HSLIDE`	

Table 3-45. Control Styles for ATTR_CTRL_STYLE (Continued)

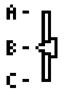
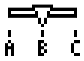
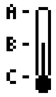
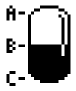
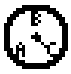
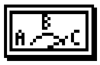
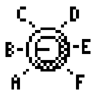
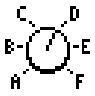
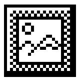
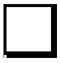
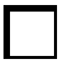
Type	Value	Icon
Rings	`CTRL_RING_POINTER_VSLIDE`	
	`CTRL_RING_POINTER_HSLIDE`	
	`CTRL_RING_THERMOMETER`	
	`CTRL_RING_TANK`	
	`CTRL_RING_GAUGE`	
	`CTRL_RING_METER`	
	`CTRL_RING_KNOB`	
	`CTRL_RING_DIAL`	
	`CTRL_PICTURE_RING`	
List box	`CTRL_LIST`	
Decorations	`CTRL_RAISED_BOX`	
	`CTRL_RECESSED_BOX`	

Table 3-45. Control Styles for ATTR_CTRL_STYLE (Continued)

Type	Value	Icon
Decorations	`CTRL_FLAT_BOX`	□
	`CTRL_RAISED_CIRCLE`	○
	`CTRL_RECESSED_CIRCLE`	○
	`CTRL_FLAT_CIRCLE`	○
	`CTRL_RAISED_FRAME`	▣
	`CTRL_RECESSED_FRAME`	▣
	`CTRL_FLAT_FRAME`	▣
	`CTRL_RAISED_ROUND_FRAME`	◎
	`CTRL_RECESSED_ROUND_FRAME`	◎
	`CTRL_FLAT_ROUND_FRAME`	◎
	`CTRL_RAISED_ROUNDED_BOX`	▢
	`CTRL_RECESSED_ROUNDED_BOX`	▢
	`CTRL_FLAT_ROUNDED_BOX`	▢
Graph	`CTRL_GRAPH`	📈

Table 3-45. Control Styles for ATTR_CTRL_STYLE (Continued)

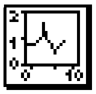
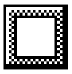
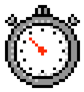
Type	Value	Icon
Strip chart	`CTRL_STRIP_CHART`	
Picture	`CTRL_PICTURE`	
Timer	`CTRL_TIMER`	
Canvas	`CTRL_CANVAS`	

Table 3-46 presents the control data types that correspond to the `ATTR_DATA_TYPE` attribute.

Table 3-46. Control Data Types for the ATTR_DATA_TYPE Attribute

Values
`VAL_CHAR`
`VAL_INTEGER`
`VAL_SHORT_INTEGER`
`VAL_FLOAT`
`VAL_DOUBLE`
`VAL_STRING`
`VAL_UNSIGNED_SHORT_INTEGER`
`VAL_UNSIGNED_INTEGER`
`VAL_UNSIGNED_CHAR`

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 3-47 contains the numeric formats you can use with ATTR_FORMAT.

Table 3-47. Numeric Formats

Numeric Formats	Example
VAL_FLOATING_PT_FORMAT	123.000
VAL_SCIENTIFIC_FORMAT	1.23E+2
VAL_ENGINEERING_FORMAT	123.00E+0
VAL_DECIMAL_FORMAT*	123
VAL_HEX_FORMAT*	7B
VAL_OCTAL_FORMAT*	173
VAL_BINARY_FORMAT*	1111011
* 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 appear in the following table.

Table 3-48. ATTR_CHECK_RANGE Values

Type of Range Checking	LabWindows/CVI Action
VAL_COERCE	The value is coerced to the upper or lower range boundary, whichever is closer.
VAL_IGNORE	The value remains unchanged.
VAL_NOTIFY	The user interface operator is notified with an Out of Range dialog box when the control is active.

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/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 programmatically change the contents of a picture control by calling `DisplayImageFile`.

Changing the Contents of the Picture Control

You can change the contents of picture controls programmatically. Call `DisplayImageFile` to change the contents of a simple picture control. Call `SetCtrlAttribute` with the `ATTR_IMAGE_FILE` attribute to change the contents of a picture command button or picture toggle button. Call `ReplaceListItem` using the image file pathname as the label parameter to change the contents of a picture ring control.

Picture Control Attributes

You create the picture command button, picture toggle button, and picture ring control under the **Create** menu, within the submenus **Command Button**, **Toggle Button**, and **Ring**. The picture versions of these controls have nearly all the features of other controls.

You can use `SetCtrlVal` 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 `SetCtrlBitmap` to select images directly from memory, instead of through a pathname to a standard image file. `SetCtrlBitmap` is an alternative to `ATTR_IMAGE_FILE`, `DisplayImageFile`, and `ReplaceListItem`. For more information, refer to 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 might 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, you do not see the color shift that normally takes place in response to a mouse click. However, you 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 one of the following fit modes when you paste the image into the control: Stick Image to Corner or Center Image. When 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

You use the following functions to draw on a canvas:

- `CanvasDrawPoint` draws a point.
- `CanvasDrawLine` draws a line.
- `CanvasDrawLineTo` draws a line from the current pen position.
- `CanvasDrawRect` draws a rectangle.
- `CanvasDimRect` overlays a checkerboard pattern in a rectangular area.
- `CanvasDrawRoundedRect` draws a rectangle with rounded corners.

- `CanvasDrawOval` draws an oval.
- `CanvasDrawArc` draws an arc.
- `CanvasDrawPoly` draws a polygon.
- `CanvasDrawText` draws text within a rectangular area.
- `CanvasDrawTextAtAPoint` draws text at an anchor point.
- `CanvasDrawBitmap` draws a bitmap image.
- `CanvasScroll` scrolls a rectangular area.
- `CanvasInvertRect` inverts the colors in a rectangular area.
- `CanvasClear` restores 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, include as many drawing primitives as possible within a batch drawing operation. When you call a drawing function outside of a batch operation, LabWindows/CVI implicitly surrounds the function call with 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. You perform all drawing relative to this coordinate system. You can modify the coordinate system 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.

Off-Screen Bitmap

Each canvas has an off-screen bitmap that LabWindows/CVI uses to restore the appearance of the canvas when the region is exposed. You can choose to draw directly to the screen, bypassing the off-screen bitmap. If you draw to the off-screen 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.

`CanvasUpdate` immediately copies the canvas off-screen 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. You use `SetCtrlAttribute` to individually set the following canvas pen attributes:

```
ATTR_PEN_WIDTH
ATTR_PEN_STYLE
ATTR_PEN_COLOR
ATTR_PEN_FILL_COLOR
ATTR_PEN_MODE
ATTR_PEN_PATTERN
```

`CanvasDefaultPen` resets all these attributes to their default values.

The location of the pen affects the starting position of the line drawn by `CanvasDrawLineTo`. The location of the pen is affected only by `CanvasSetPenPosition` and `CanvasDrawLineTo`. 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 off-screen 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

Table 3-49 lists the attributes for Canvas Controls.

Table 3-49. Control Attributes for Canvas Controls

Name	Type	Description
`ATTR_DRAW_POLICY`	integer	Determines when drawing operations are rendered on the off-screen bitmap and the screen. Refer to discussion that follows this table.
`ATTR_OVERLAPPED_POLICY`	integer	Determines what occurs when you draw to a canvas which is overlapped by another control. Refer to discussion that follows this table.
`ATTR_PEN_COLOR`	integer	RGB color value used to draw points, lines, frames, and text on the canvas.
`ATTR_PEN_FILL_COLOR`	integer	RGB color value used to fill interior areas of shapes, text backgrounds, and areas exposed by scrolling.
`ATTR_PEN_MODE`	integer	Determines the effect of drawing with the pen color or pen fill color, given the current color on the screen. Refer to discussion following this table.
`ATTR_PEN_PATTERN`	unsigned char[8]	Determines the pattern used to fill interior areas of shapes. Refer to discussion following this table.
`ATTR_PEN_STYLE`	integer	Style used when drawing lines and frames. Under Windows, applies only if pen width is 1; if pen width is greater than 1, the style is always `VAL_SOLID`. Refer to Table 3-64.
`ATTR_PEN_WIDTH`	integer	Number of pixels in the width of a pen stroke. Applies to lines, frames, and points. Valid range: 1 to 255. Default: `1`.

Table 3-49. Control Attributes for Canvas Controls (Continued)

Name	Type	Description
`ATTR_XCOORD_AT_ORIGIN`	double	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`.
`ATTR_XSCALING`	double	Factor used to scale your horizontal coordinates and widths into pixel-based coordinates and widths. Default value: `1.0`.
`ATTR_YCOORD_AT_ORIGIN`	double	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`.
`ATTR_YSCALING`	double	Factor used to scale your vertical coordinates and heights into pixel-based coordinates and heights. Default value: `1.0`.

Table 3-50 lists the values associated with the `ATTR_DRAW_POLICY` attribute.

Table 3-50. Values for ATTR_DRAW_POLICY

Value	Description
`VAL_UPDATE_IMMEDIATELY`	Drawing takes place off screen. The section of the bitmap corresponding to the area of the drawing operation is copied into the canvas display immediately. This is the default behavior.
`VAL_MARK_FOR_UPDATE`	Drawing takes place off-screen. 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.
`VAL_DIRECT_TO_SCREEN`	Drawing goes directly to the screen. The off-screen bitmap is not updated. Although this might result in a faster drawing time, whatever is drawn in this mode is lost when the canvas is redrawn.

Table 3-51 lists the values associated with the `ATTR_OVERLAPPED_POLICY` attribute.

Table 3-51. Values for ATTR_OVERLAPPED_POLICY

Value	Description
`VAL_DEFER_DRAWING`	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 behavior.
`VAL_DRAW_ON_TOP`	If the control is overlapped and the draw policy is *not* `VAL_MARK_FOR_UPDATE`, drawing occurs on top of the overlapping controls.

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. When you use the default setting, `VAL_COPY_MODE`, the pen color or pen fill color replaces the current screen 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 affect the palette indices rather than the RGB values, depending on the operating system.***

Table 3-52 lists the values associated with the `ATTR_PEN_MODE` attribute.

Table 3-52. Values for ATTR_PEN_MODE

Value	Description
`VAL_COPY_MODE`	pen color (the default)
`VAL_OR_MODE`	pen color | screen color
`VAL_XOR_MODE`	pen color ^ screen color
`VAL_AND_NOT_MODE`	~(pen color) and screen color
`VAL_NOT_COPY_MODE`	~(pen color)
`VAL_OR_NOT_MODE`	~(pen color) | screen color
`VAL_NOT_XOR_MODE`	~(pen color ^ screen color)
`VAL_AND_MODE`	pen color and screen color

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," perform the following steps:

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

The userint.h include file defines two structures, Rect and Point. You use these structures to specify locations and areas in Cartesian coordinate systems, such as those 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:

```
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:

```
typedef struct
   {
   int x;
   int y;
   } Point;
```

Functions and Macros for Making Rects and Points

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

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

`MakePoint` creates a `Point` in the following example:

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

You can also use these functions to initialize variables, as in the following example:

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

The `Rect` height and width have special values. Also, some macros exist for creating commonly used rectangles. The documentation for each function indicates when these values and macros are applicable. Refer to Table 3-53.

Table 3-53. Values and Macros for Rect Structures

Name	Value or Definition	Description
`VAL_TO_EDGE`	`–1`	Set the `Rect` width (or height) to the distance from the `Rect` left (or top) to the right (or bottom) edge of the object.
`VAL_KEEP_SAME_SIZE`	`–2`	When copying objects such as bitmaps, make the destination object the same size as the source object.
`VAL_EMPTY_RECT`	`MakeRect (0, 0, 0, 0)`	An empty rectangle.
`VAL_ENTIRE_OBJECT`	`MakeRect (0, 0, VAL_TO_EDGE, VAL_TO_EDGE)`	Make the `Rect` the size of the object, for example, the canvas or bitmap.

Functions for Modifying Rects and Points

You use the following functions to set or modify the values in a `Rect` or `Point` structure:

- `RectSet` sets each of the four values of an existing `Rect` structure.
- `RectSetFromPoints` sets a `Rect` so that it defines the smallest rectangle that encloses two points.
- `RectSetBottom` sets the height of a `Rect` so that the bottom is a given value. The bottom is *not* enclosed by the rectangle.
- `RectSetRight` sets the height of a `Rect` so that the right edge is a given value. The right edge is *not* enclosed by the rectangle.
- `RectSetCenter` sets the top and left of a `Rect` so that it is centered around a given value, while keeping the same size.
- `RectOffset` modifies the top and left of a `Rect` so as to shift the location of the rectangle.
- `RectMove` sets the top and left of `Rect` to a given `Point`.
- `RectGrow` modifies the values in a `Rect` so that the rectangle grows or shrinks around its current center point.
- `PointSet` sets the two values in an existing `Point` structure.

Functions for Comparing or Obtaining Values from Rects and Points

You use the following functions to compare or obtain values from a `Rect` or `Point` structure:

- `RectBottom` obtains the location of the bottom of a rectangle.
- `RectRight` obtains the location of the right edge of a rectangle.
- `RectCenter` obtains the location of the center of a rectangle.
- `RectEqual` determines if two rectangles are identical.
- `RectEmpty` determines if a rectangle is empty.
- `RectContainsPoint` determines if a rectangle encloses a given point.
- `RectContainsRect` determines if a rectangle completely encloses another rectangle.
- `RectSameSize` determines if two rectangles are the same size.
- `RectUnion` sets a `Rect` to the smallest rectangle that encloses two given rectangles.
- `RectIntersection` sets a `Rect` to the largest rectangle that is enclosed by two given rectangles.
- `PointEqual` determines whether two points are at the same location.
- `PointPinnedToRect` modifies a `Point` structure, if necessary, to ensure that it is within a given rectangle.

Using Bitmap Objects

A bitmap is a two-dimensional grid of pixels representing an image. Some functions exist, 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.

Other functions exist, however, that 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

You use the following functions to create, extract, or discard bitmap objects:

- `NewBitmap` creates a bitmap object from scratch.
- `GetBitmapFromFile` creates a bitmap object using image data from a file.
- `GetCtrlBitmap` creates a bitmap object from an image contained in a picture, picture ring, picture button, canvas, or graph control.
- `GetCtrlDisplayBitmap` creates a bitmap object from the current appearance of a control.
- `GetScaledCtrlDisplayBitmap` creates a bitmap object from the current appearance of a control and scales it to a specific rectangular area.
- `GetPanelDisplayBitmap` creates a bitmap object from the current appearance of a specific rectangular area of a panel.
- `GetScaledPanelDisplayBitmap` creates a bitmap object from the current appearance of a specific rectangular area of a panel and scales it to a different rectangular area.
- `ClipboardGetBitmap` creates a bitmap object from an image, if any, in the system clipboard.
- `DiscardBitmap` removes 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 `NewBitmap`.

Windows Metafiles

A Windows metafile (`.wmf`) contains a description of an image that is scalable without distortion. The description consists of a set of drawing commands rather than a bitmap. Windows metafiles are available only under Windows 95/NT. You can load them using `GetBitmapFromFile`.

Functions for Displaying or Copying Bitmap Objects

You use the following functions to display a bitmap object in a control or copy an image from a bitmap object to a control:

- `CanvasDrawBitmap` displays a bitmap in a canvas control.
- `SetCtrlBitmap` sets an image in a picture, picture ring, picture button, or graph control from a bitmap object. You can use it to replace an existing image or to create a new image.
- `SetBitmapData` changes the contents of an existing bitmap. This approach is faster than discarding a bitmap and creating a new one.
- `ClipboardPutBitmap` copies image data from a bitmap object to the system clipboard.

Functions for Retrieving Image Data from Bitmap Objects

You use the following functions to retrieve image data from bitmap objects:

- `GetBitmapInfo` obtains size information about the image associated with a bitmap. You then use this information to allocate the buffers you pass to `GetBitmapData`.
- `AllocBitmapData` allocates the buffers necessary for calling `GetBitmapData`. This is an alternative to calling `GetBitmapInfo` and allocating the buffers yourself.
- `GetBitmapData` obtains 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`. The number of timer controls you can use is unlimited, 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. You use the following functions to manage timer controls:

`SuspendTimerCallbacks` stops calls to all the timer control callbacks until you call `ResumeTimerCallbacks` to restore 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.

`ResetTimer` resets 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.

```
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 3-54 lists the timer control attributes you can retrieve or change using `SetCtrlAttribute` and `GetCtrlAttribute`.

Table 3-54. Timer Control Attributes

Name	Type	Description
`ATTR_INTERVAL`	double	Time interval of the timer control in seconds.
`ATTR_ENABLED`	integer	`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 LabWindows/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 LabWindows/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 might 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 might 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, refer to the *Programming with Controls* section of this chapter. You also use `GetCtrlAttribute` and `SetCtrlAttribute` to control the attributes of graphs and strip charts. You use the following functions to programmatically change the axis ranges of graphs and strip charts and change the scaling mode:

`GetAxisScalingMode` obtains the current scaling mode and range of one of the axes of a graph or strip chart control.

`SetAxisScalingMode` sets the current scaling mode and range of one of the axes of a graph or the y-axis of a strip chart. You cannot set the minimum or maximum value for a strip chart x-axis.

`GetCtrlAttribute` or `SetCtrlAttribute` with the `ATTR_XAXIS_OFFSET` and `ATTR_XAXIS_GAIN` attributes obtains or modifies the x-offset and x-increment for a strip chart.

Functions for Graphs Only

You can use one of the following functions to add a plot to a graph:

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

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

`DeleteGraphPlot` removes one or all plots from a graph. `DeleteGraphPlot` frees specific plots from memory.

`GetPlotAttribute` obtains an attribute of a particular graph plot.

`SetPlotAttribute` sets an attribute of a particular graph plot.

You can use one of the following functions to control the cursors in a graph:

- `GetGraphCursor` obtains the current position of a specified cursor.
- `SetGraphCursor` sets the position of a specified cursor.
- `GetGraphCursorIndex` obtains the plot handle and the array index of the plot that the specified cursor is attached to.
- `SetGraphCursorIndex` attaches a cursor to a particular data point.
- `GetActiveGraphCursor` obtains the index of the active graph cursor.
- `SetActiveGraphCursor` sets the active graph cursor.

- `GetCursorAttribute` obtains an attribute of a specified cursor.
- `SetCursorAttribute` sets an attribute of a specified cursor.

Functions for Strip Charts Only

You use the following functions to control strip charts only:

- `PlotStripChart` adds 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.
- `PlotStripChartPoint` adds one point to a strip chart that contains exactly one trace.
- `ClearStripChart` clears all points from the strip chart. If the strip chart is visible, the points on the plot clear.
- `GetTraceAttribute` obtains an attribute of a particular strip chart trace
- `SetTraceAttribute` sets an attribute of a particular strip chart trace.

Processing Graph and Strip Chart Events

Graph events are processed like any other control. Refer to 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. Refer to the *Processing Control Events* section of this chapter for details about processing events using callbacks.

Graph and Strip Chart Attributes

Tables 3-55 to 3-60 list control attributes you can access through `GetCtrlAttribute` and `SetCtrlAttribute`.

Table 3-55. Graph and Strip Chart Attributes

Name	Type	Description
`ATTR_BORDER_VISIBLE`	integer	`1` = border visible `0` = border invisible
`ATTR_EDGE_STYLE`	integer	`VAL_RAISED_EDGE` or `VAL_FLAT_EDGE` or `VAL_RECESSED_EDGE`
`ATTR_GRAPH_BGCOLOR`	integer	Graph border background color. RGB value. Refer to discussion that follows this table.
`ATTR_GRID_COLOR`	integer	Grid color. RGB value. Refer to discussion that follows this table.

Table 3-55. Graph and Strip Chart Attributes (Continued)

Name	Type	Description
`ATTR_INNER_LOG_MARKERS_VISIBLE`	integer	Specifies whether labels and tick marks appear next to the inner grid lines of log scale axes. Default value: FALSE.
`ATTR_PLOT_AREA_HEIGHT`	integer	Height of the plotting area in pixels (`GetCtrlAttribute` only).
`ATTR_PLOT_AREA_LEFT`	integer	Offset in pixels of the left edge of the plot area from the left edge of the control. (`GetCtrlAttribute` only).
`ATTR_PLOT_AREA_TOP`	integer	Offset in pixels of the top of the plot area from the top of the control. (`GetCtrlAttribute` only).
`ATTR_PLOT_AREA_WIDTH`	integer	Width of the plotting area in pixels (`GetCtrlAttribute` only).
`ATTR_PLOT_BGCOLOR`	integer	Plot background color. RGB value. Refer to discussion that follows this table.
`ATTR_XAXIS_GAIN`	double	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`.
`ATTR_XAXIS_OFFSET`	double	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`.
`ATTR_XDIVISIONS`	integer	1 to 100, or `VAL_AUTO`
`ATTR_XENG_UNITS`	integer	–308 to 308
`ATTR_XFORMAT`	integer	Refer to Table 3-47.
`ATTR_XGRID_VISIBLE`	integer	`1` = x-grid visible `0` = x-grid invisible

Table 3-55. Graph and Strip Chart Attributes (Continued)

Name	Type	Description
ATTR_XLABEL_VISIBLE	integer	1 = x-label visible 0 = x-label invisible
ATTR_XNAME	char *	Name of x-axis.
ATTR_XUSE_LABEL_STRINGS	integer	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 InsertAxisItem.
ATTR_YAXIS_GAIN	double	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.
ATTR_YAXIS_OFFSET	double	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.
ATTR_Y_REVERSE	integer	Whether to reverse the orientation of the y-axis so that the lowest value appears at the top. If the orientation of the y-axis is reversed, the vertical orientation of the plots is also reversed.
ATTR_YUSE_LABEL_STRINGS	integer	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 InsertAxisItem.
ATTR_XNAME_LENGTH	integer	Number of characters in x-axis name (GetCtrlAttribute only).
ATTR_XPRECISION	integer	0 to 15, or VAL_AUTO

Table 3-55. Graph and Strip Chart Attributes (Continued)

Name	Type	Description
`ATTR_XYLABEL_BOLD`	integer	`1` = x- and y-axis labels bold `0` = x- and y-axis labels not bold
`ATTR_XYLABEL_COLOR`	integer	x- and y-axis label color. RGB value. Refer to discussion that follows this table.
`ATTR_XYLABEL_FONT`	char *	x- and y-axis label font. Refer to Table 3-5.
`ATTR_XYLABEL_FONT_NAME_LENGTH`	integer	Number of characters in font of x- and y-axis labels (`GetCtrlAttribute` only).
`ATTR_XYLABEL_ITALIC`	integer	`1` = x- and y-axis labels in italics `0` = x- and y-axis labels not in italics
`ATTR_XYLABEL_POINT_SIZE`	integer	Point size of x- and y-axis names; range = 0 to 32,767.
`ATTR_XYLABEL_STRIKEOUT`	integer	`1` = x- and y-axis labels have strikeout `0` = x- and y-axis labels do not have strikeout
`ATTR_XYLABEL_UNDERLINE`	integer	`1` = x- and y-axis labels underlined `0` = x- and y-axis labels not underlined
`ATTR_XYNAME_BOLD`	integer	`1` = x- and y-axis names bold `0` = x- and y-axis names not bold
`ATTR_XYNAME_COLOR`	integer	x- and y-axis name color. RGB value. Refer to discussion that follows this table.
`ATTR_XYNAME_FONT`	char *	x- and y-axis name font. Refer to Table 3-5.
`ATTR_XYNAME_FONT_NAME_LENGTH`	integer	Number of characters in font of x- and y-axis names (`GetCtrlAttribute` only).
`ATTR_XYNAME_ITALIC`	integer	`1` = x- and y-axis names in italics `0` = x- and y-axis names not in italics
`ATTR_XYNAME_POINT_SIZE`	integer	Point size of x- and y-axis names range = 0 to 32,767.

Table 3-55. Graph and Strip Chart Attributes (Continued)

Name	Type	Description
`ATTR_XYNAME_STRIKEOUT`	integer	`1` = x- and y-axis names have strikeout `0` = x- and y-axis names do not have strikeout
`ATTR_XYNAME_UNDERLINE`	integer	`1` = x- and y-axis names underlined `0` = x- and y-axis names not underlined
`ATTR_YDIVISIONS`	integer	Range: 1 to 100, or `VAL_AUTO`
`ATTR_YENG_UNITS`	integer	Range: –308 to 308
`ATTR_YFORMAT`	integer	Refer to Table 3-47.
`ATTR_YGRID_VISIBLE`	integer	`1` = y-grid visible `0` = y-grid invisible
`ATTR_YLABEL_VISIBLE`	integer	`1` = y-label visible `0` = y-label invisible
`ATTR_YMAP_MODE`	integer	`VAL_LINEAR` or `VAL_LOG`
`ATTR_YNAME`	char *	y-axis name
`ATTR_YNAME_LENGTH`	integer	Number of characters in y-axis name (`GetCtrlAttribute` only).
`ATTR_YPRECISION`	integer	Range: 0 to 15, or `VAL_AUTO`

Table 3-56. Graph Attributes

Name	Type	Description
`ATTR_ACTIVE_YAXIS`	integer	Which of the two y-axes is used in plotting, setting a y-axis attribute, setting the y-axis range, or creating a graph cursor. Values: `VAL_LEFT_YAXIS`, `VAL_RIGHT_YAXIS`.
`ATTR_COPY_ORIGINAL_DATA`	integer	Refer to discussion below this table.
`ATTR_DATA_MODE`	integer	`VAL_RETAIN` or `VAL_DISCARD`. Refer to discussion that follows this table.

Table 3-56. Graph Attributes (Continued)

Name	Type	Description
`ATTR_ENABLE_ZOOMING`	integer	Whether the end-user can interactively zoom and pan the graph view port. Default: FALSE
`ATTR_NUM_CURSORS`	integer	Number of cursors (0 to 10).
`ATTR_REFRESH_GRAPH`	integer	`1` = plot to screen immediately `0` = do not plot to screen until graph you set `ATTR_REFRESH_GRAPH` to `1` or you call `RefreshGraph`.
`ATTR_SHIFT_TEXT_PLOTS`	integer	`1` = text is shifted so it is not clipped `0` = text might be clipped
`ATTR_SMOOTH_UPDATE`	integer	`1` = use off-screen bitmap `0` = do not use off-screen bitmap Refer to discussion that follows this table.
`ATTR_XMAP_MODE`	integer	`VAL_LINEAR` or `VAL_LOG`
`ATTR_XMARK_ORIGIN`	integer	`1` = tick marks along x-origin `0` = no tick marks along x-origin
`ATTR_XREVERSE`	integer	Whether to reverse the orientation of the x-axis so that the lowest value appears at the right. If the orientation of the x-axis is reversed, the horizontal orientation of the plots is also reversed.
`ATTR_YMARK_ORIGIN`	integer	`1` = tick marks along y-origin `0` = no tick marks along y-origin

Table 3-57. Strip Chart Attributes

Name	Type	Description
`ATTR_NUM_TRACES`	integer	Number of traces (1 to 64).
`ATTR_POINTS_PER_SCREEN`	integer	Range: 3 to 10,000
`ATTR_SCROLL_MODE`	integer	`VAL_SWEEP` or `VAL_CONTINUOUS` or `VAL_BLOCK` Refer to discussion that follows this table.

Table 3-58. Graph Cursor Attributes (GetCursorAttribute and SetCursorAttribute)

Name	Type	Description
`ATTR_CROSS_HAIR_STYLE`	integer	Refer to Table 3-61.
`ATTR_CURSOR_COLOR`	integer	Cursor color. RGB value. Refer to discussion that follows this table.
`ATTR_CURSOR_MODE`	integer	`VAL_FREE_FORM` or `VAL_SNAP_TO_POINT` Refer to discussion that follows this table.
`ATTR_CURSOR_POINT_STYLE`	integer	Refer to Table 3-62.
`ATTR_CURSOR_YAXIS`	integer	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`.

Table 3-59. Graph Plot (GetPlotAttribute and SetPlotAttribute) and Strip Chart Trace (GetTraceAttribute and SetTraceAttribute) Attributes

Name	Type	Description
`ATTR_LINE_STYLE`	integer	Refer to Table 3-63.
`ATTR_PLOT_STYLE`	integer	Refer to Table 3-64.
`ATTR_TRACE_COLOR`	integer	Trace color. RGB value. Refer to discussion that follows this table.
`ATTR_TRACE_POINT_STYLE`	integer	Refer to Table 3-62.
`ATTR_TRACE_VISIBLE`	integer	`1` = trace is visible `0` = trace is invisible

Table 3-60. Graph Plot Attributes (GetPlotAttribute and SetPlotAttribute)

Name	Type	Description
`ATTR_INTERPOLATE_PIXELS`	integer	Enable the calculation of the displayed color for each pixel by using interpolation. Valid for Intensity plots only. `0` = Interpolate pixels `1` = No Interpolation
`ATTR_NUM_POINTS`	integer	Number of points in the plot data (`GetPlotAttribute` only). For intensity plots, the number of points is equal to the number of points in the two-dimensional z-data array. Valid for x, y, xy, Waveform, Polygon, and Intensity plots only.
`ATTR_PLOT_FONT`	char *	Font name for the text plot. Valid for `PlotText` plots only.
`ATTR_PLOT_FONT_NAME_LENGTH`	integer	Length of the font name for the text plot (`GetPlotAttribute` only). Valid for `PlotText` plots only.
`ATTR_PLOT_ORIGIN`	integer	Determines the placement of a text string or bitmap with respect to the coordinates specified in a call to `PlotText` or `PlotBitmap`. Refer to discussion following this table.
`ATTR_PLOT_SNAPPABLE`	integer	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.

Table 3-60. Graph Plot Attributes (GetPlotAttribute and SetPlotAttribute) (Continued)

Name	Type	Description
`ATTR_PLOT_YAXIS`	integer	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`.
`ATTR_PLOT_ZPLANE_POSITION`	integer	Drawing order of the graph plot. The lowest ordered plot (`0`) is on top. Valid Range: 0 to (Number of Plots –1)
`ATTR_TRACE_BGCOLOR`	integer	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.
`ATTR_PLOT_THICKNESS`	integer	Thickness of the plot line, in pixels. Applies only when `ATTR_LINE_STYLE` is `VAL_SOLID`. If `ATTR_PLOT_STYLE` is `ATTR_FAT_LINE` or `ATTR_FAT_STEP`, the plot has three times the thickness you specify in this attribute. Range: `1` to `32`. The default is `1`.
`ATTR_PLOT_XDATA`	void *	A void pointer to a buffer for the x data to be copied to (`GetPlotAttribute` only). Valid for x-, xy-, and Polygon plots only.
`ATTR_PLOT_YDATA`	void *	A void pointer to a buffer for the y data to be copied to (`GetPlotAttribute` only). Valid for y-, xy-, Waveform, and Polygon plots only.
`ATTR_PLOT_ZDATA`	void *	A void pointer to a buffer for the *z* data to be copied to (`GetPlotAttribute` only). The z-data is a two-dimensional array. Valid for Intensity plots only.

Table 3-60. Graph Plot Attributes (GetPlotAttribute and SetPlotAttribute) (Continued)

Name	Type	Description
`ATTR_PLOT_XDATA_TYPE`	integer	Type of data in the x data plot (`GetPlotAttribute` only). Valid for x-, xy-, and Polygon plots only.
`ATTR_PLOT_YDATA_TYPE`	integer	Type of data in the y data plot (`GetPlotAttribute` only). Valid for y-, xy-, Waveform, and Polygon plots only.
`ATTR_PLOT_ZDATA_TYPE`	integer	Type of data in the z data plot (`GetPlotAttribute` only). Valid for Intensity plots only.
`ATTR_PLOT_XDATA_SIZE`	integer	Number of bytes in the x data plot data (`GetPlotAttribute` only). Valid for x-, xy-, and Polygon plots only.
`ATTR_PLOT_YDATA_SIZE`	integer	Number of bytes in the y data plot data (`GetPlotAttribute` only). Valid for y-, xy-, Waveform, and Polygon plots only.
`ATTR_PLOT_ZDATA_SIZE`	integer	Number in bytes in the z data plot data (`GetPlotAttribute` only). Valid for Intensity plots only.

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. Refer to Table 3-3 for a list of common color values.

The `ATTR_DATA_MODE` attribute controls whether LabWindows/CVI keeps or discards scaled plot data after drawing new plots 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 or when you print the graph. 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 when the graph is rescaled or when the graph is redrawn while `ATTR_SMOOTH_UPDATE` is disabled. The plots do not appear when you print unless you enable bitmap printing and smooth updates. Also, when `ATTR_DATA_MODE` is `VAL_DISCARD`, the plotting functions do not return plot handles.

The ATTR_COPY_ORIGINAL_DATA attribute specifies whether LabWindows/CVI makes a copy of your original plot data or merely retains a pointer to it. LabWindows/CVI uses the original plot data whenever it rescales the graph. If LabWindows/CVI keeps a pointer to your data and you change the data or deallocate the array, erroneous results occur. You can prevent this problem by setting ATTR_COPY_ORIGINAL_DATA to TRUE, but then LabWindows/CVI uses more memory and time whenever you add a plot to the graph.

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 attribute 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 3-61 shows the cursor styles associated with ATTR_CROSS_HAIR_STYLE. Assume that ATTR_CURSOR_POINT_STYLE is set to VAL_SIMPLE_DOT.

Table 3-61. Cursor Styles for ATTR_CROSS_HAIR_STYLE

Value	Cross Hair Style
VAL_LONG_CROSS	
VAL_VERTICAL_LINE	
VAL_HORIZONTAL_LINE	
VAL_NO_CROSS	
VAL_SHORT_CROSS	

Table 3-62 shows the styles associated with `ATTR_CURSOR_POINT_STYLE` and `ATTR_TRACE_POINT_STYLE`.

Table 3-62. Styles for ATTR_CURSOR_POINT_STYLE and ATTR_TRACE_POINT_STYLE

Value Style Name	Cursor/Point Style
`VAL_EMPTY_SQUARE`	□
`VAL_SOLID_SQUARE`	■
`VAL_ASTERISK`	✱
`VAL_DOTTED_EMPTY_SQUARE`	⊡
`VAL_DOTTED_SOLID_SQUARE`	▣
`VAL_SOLID_DIAMOND`	◆
`VAL_EMPTY_SQUARE_WITH_X`	⊠
`VAL_EMPTY_SQUARE_WITH_CROSS`	⊞
`VAL_BOLD_X`	✖
`VAL_SMALL_SOLID_SQUARE`	▪
`VAL_SIMPLE_DOT`	·
`VAL_EMPTY_CIRCLE`	○
`VAL_SOLID_CIRCLE`	●
`VAL_DOTTED_SOLID_CIRCLE`	◉
`VAL_DOTTED_EMPTY_CIRCLE`	⊙
`VAL_BOLD_CROSS`	✚
`VAL_CROSS`	+
`VAL_SMALL_CROSS`	﹢
`VAL_X`	✕
`VAL_SMALL_X`	×

Table 3-62. Styles for ATTR_CURSOR_POINT_STYLE and ATTR_TRACE_POINT_STYLE (Continued)

Value Style Name	Cursor/Point Style
`VAL_DOTTED_SOLID_DIAMOND`	◆
`VAL_EMPTY_DIAMOND`	◇
`VAL_DOTTED_EMPTY_DIAMOND`	◈
`VAL_SMALL_EMPTY_SQUARE`	▫
`VAL_NO_POINT`	

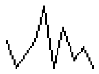
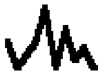
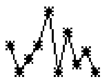
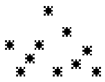
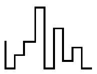
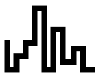
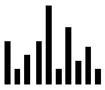
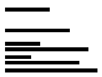
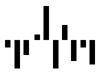
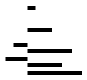
Table 3-63 shows the line styles associated with `ATTR_LINE_STYLE`. Assume that `ATTR_PLOT_STYLE` is set to `VAL_THIN_LINE`.

Table 3-63. Line Styles for ATTR_LINE_STYLE

Value	Line Style
`VAL_SOLID`	╱
`VAL_DASH`	⁄ ⁄
`VAL_DOT`	⋰
`VAL_DASH_DOT`	⁄ · ⁄
`VAL_DASH_DOT_DOT`	⁄ · · ⁄

Table 3-64 shows the plot styles associated with ATTR_PLOT_STYLE. Assume that the point style is set to VAL_ASTERISK.

Table 3-64. Plot Styles for ATTR_PLOT_STYLE

Value	Plot Style
VAL_THIN_LINE	
VAL_FAT_LINE	
VAL_CONNECTED_POINTS	
VAL_SCATTER	
VAL_THIN_STEP	
VAL_FAT_STEP	
VAL_VERTICAL_BAR (not valid for strip charts)	
VAL_HORIZONTAL_BAR (not valid for strip charts)	
VAL_BASE_ZERO_VERTICAL_BARS (not valid for strip charts)	
VAL_BASE_ZERO_HORIZONTAL_BARS (not valid for strip charts)	

☞ **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 appear in Table 3-65.

Table 3-65. Values for ATTR_PLOT_ORIGIN

Value	Description
`VAL_LOWER_LEFT`	Lower left corner of the enclosing rectangle.
`VAL_CENTER_LEFT`	Midpoint of the left edge of the enclosing rectangle.
`VAL_UPPER_LEFT`	Upper left corner of the enclosing rectangle.
`VAL_LOWER_CENTER`	Midpoint of the bottom edge of the enclosing rectangle.
`VAL_CENTER_CENTER`	Center of the enclosing rectangle.
`VAL_UPPER_CENTER`	Midpoint of the top edge of the enclosing rectangle.
`VAL_LOWER_RIGHT`	Lower right corner of the enclosing rectangle.
`VAL_CENTER_RIGHT`	Midpoint of the right edge of the enclosing rectangle.
`VAL_UPPER_RIGHT`	Upper right corner of the enclosing rectangle.

Two Y-Axes (Graphs Only)

A graph always contains two y-axes. By default, only the left y-axis is visible.

You can make the y-axis visible by using the following code:

```
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.)

After you have added a plot a graph, you can associate it with the other y-axis by using the `ATTR_PLOT_YAXIS` attribute.

After 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 attributes discussed in this section 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 the graph updates to the off-screen buffer before plotting to the screen. However, when you move or close a window that covers the graph, `ATTR_SMOOTH_UPDATE` lets LabWindows/CVI refresh the graph quickly because the graph already exists in the off-screen buffer.

Speed and VAL_AUTO_SCALE

When you enable autoscaling, LabWindows/CVI recalculates the limits of the axes and remaps the existing graph plots with every new plot. As the number of existing plots on a graph increases, try to minimize scaling adjustments because the time necessary to recalculate and remap all the existing plots increases. You can minimize delays by disabling autoscaling, thereby preventing the recalculation and re-mapping of the graph plots. To disable autoscaling, call `SetAxisRange` with the **xAxisScaling** and the **yAxisScaling** parameters each set to `VAL_MANUAL` or `VAL_LOCK`.

Controlling How Graphs Refresh

You can use `RefreshGraph`, the `ATTR_REFRESH_GRAPH` attribute, and the **refresh** parameter of `DeleteGraphPlot` to tell LabWindows/CVI when to update the graph. When you call `DeleteGraphPlot`, you choose whether to redraw the plot area immediately (`VAL_IMMEDIATE_DRAW`), not redraw it at all (`VAL_NO_DRAW`), or mark the plot area to be redrawn later (`VAL_DELAYED_DRAW`). If you pass `VAL_DELAYED_DRAW` for the **refresh** parameter, LabWindows/CVI does not redraw the plot area until one of the following conditions occurs:

- You expose the plot area after hiding or overlapping it.
- You change almost any graph attribute.
- You add a new plot while `ATTR_REFRESH_GRAPH` is enabled.
- You call `DeleteGraphPlot` with the **refresh** parameter set to `VAL_IMMEDIATE_DRAW.`
- You enable `ATTR_REFRESH_GRAPH` through `SetCtrlAttribute`.
- You move the cursor.
- You call `RefreshGraph`.
- You rescale the graph.

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 LabWindows/CVI draws new plots on a graph control immediately. When your program builds a multi-plot image for which only the final appearance is of interest, disable `ATTR_REFRESH_GRAPH` before you plot, and enable it after you finish plotting. That way, LabWindows/CVI redraws the graph only one time, instead of each time you add a 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. Pending plots only appear after a call to `RefreshGraph` or when you enable the `ATTR_REFRESH_GRAPH` attribute.

Optimizing Memory Usage

You can optimize memory usage by disabling the `ATTR_COPY_ORIGINAL_DATA` attribute and setting `ATTR_DATA_MODE` to `VAL_DISCARD`, 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 new plots you add. Changing the attribute does not affect existing plots. You can save memory by setting `ATTR_DATA_MODE` to `VAL_DISCARD`, but that approach has the following disadvantages:

- When you *disable* `ATTR_SMOOTH_UPDATE` and set `ATTR_DATA_MODE` to discard, you get the following results:
  - The plot does not appear on the printout when you call `PrintPanel` or `PrintCtrl`.
  - The plot disappears from the screen when the graph is redrawn for any reason.
- When you *enable* `ATTR_SMOOTH_UPDATE` and set `ATTR_DATA_MODE` to discard, you get the following results:
  - The plot does not appear on the printout when you call `PrintPanel` or `PrintCtrl` with `ATTR_BITMAP_PRINTING` set to `FALSE`.
  - The plot disappears from the screen when the graph is rescaled.

Your graph can rescale for the following reasons:

- You call `SetAxisScalingMode` or `SetAxisRange`.
- You change `ATTR_XMAP_MODE` or `ATTR_YMAP_MODE`.
- You add 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.

☞ **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 plot handles. The graph behaves as if the plot does not exist.***

The `ATTR_COPY_ORIGINAL_DATA` graph attribute determines whether the graph control keeps its own copy of the plot data. When disabled, the graph control maintains a pointer to the original array data you originally passed to the plotting function. If you change the original data or deallocate the array, either the graph displays incorrect data or an invalid pointer error occurs. `ATTR_COPY_ORIGINAL_DATA` affects only graph plots that are associated with arrays, such as `PlotWaveform` and `PlotXY`. The graph control uses the original data only when the graph is rescaled.

Note `ATTR_COPY_ORIGINAL_DATA` ***is only valid if*** `ATTR_DATA_MODE` ***is set to retain data.***

Using the System Attributes

You set and get the system attributes using the `SetSystemAttribute` and `GetSystemAttribute` functions. The system attributes 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 3-66. System Attributes

Attribute	Type	Notes
`ATTR_ALLOW_UNSAFE_TIMER_EVENTS`	integer	`1` = Allow unsafe timer events. `0` = Do not allow unsafe time events (the default) Windows 95/NT only. Refer to discussion following this table.
`ATTR_ALLOW_MISSING_CALLBACKS`	integer	`1` = `LoadPanel` and `LoadMenuBar` return a valid panel or menu bar handle even if they cannot find all callback functions referenced in the `.uir` file. `0` = `LoadPanel` and `LoadMenuBar` return an error code if they cannot find all callback functions referenced in the `.uir` file (the default). The User Interface Library keeps only one value for this attribute regardless of which thread is active in your program.
`ATTR_REPORT_LOAD_FAILURE`	integer	`1` = Display a message when `LoadPanel` or `LoadMenuBar` fail (the default). `0` = Do not display a message when `LoadPanel` or `LoadMenuBar` fail. Refer to discussion following this table.

Table 3-66. System Attributes (Continued)

Attribute	Type	Notes
`ATTR_RESOLUTION_ADJUSTMENT`	integer	Specifies degree of scaling for panels and their contents on screens with differing resolutions. Values: `0` to `100` (percentage) or `VAL_USE_PANEL_SETTING`. Refer to discussion following this table.
`ATTR_SUPPRESS_EVENT_PROCESSING`	integer	`1` = No events are processed. `0` = Events are processed normally (the default). Refer to discussion following this table.
`ATTR_TASKBAR_BUTTON_VISIBLE`	integer	`0` = Do not display a taskbar button (the default for DLLs). `1` = Display a taskbar button (the default for executables). Refer to discussion following this table.
`ATTR_TASKBAR_BUTTON_TEXT`	integer	Text to display in the taskbar button. Accepts only the first 80 characters. When calling `GetSystemAttribute`, you must pass a buffer with at least 81 bytes. Refer to discussion following this table.

Unsafe Timer Events

By default, timer control callbacks do not occur under Windows while you are moving or sizing a window, while the system menu is pulled down, or while you press <Alt-Tab>. (These conditions are called *event-blocking conditions*.) Under Windows 95/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).

This feature has several limitations. 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 after the timer callback returns.

This problem is inherent to Windows and occurs regardless of the development environment.

The User Interface Library keeps only one value for this attribute regardless of which thread is active in your program.

You should not enable this attribute until you have thoroughly debugged the code in your timer callbacks. 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 might not be able to find the `.uir` file. Or a callback function specified in the panel or menu bar might not be defined in the project, executable, or DLL.

It is not unusual for a load function 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 both in effect. The attribute is enabled by default.

If a load function fails in a stand-alone executable or DLL created in an external compiler, the most common reasons are:

- The executable is in a directory different from the one in which you last saved the LabWindows/CVI project file. 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. (Refer to the **External Compiler Support** command in the **Build** menu of the Project window.) This oversight causes 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 oversight causes 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 oversight causes the library to fail to find callback functions referenced in the `.uir` file.

The User Interface Library keeps only one value for this attribute regardless of which thread is active in your program.

Resolution Adjustment

When a panel is displayed on a screen with a different resolution than the screen on which you edited the panel, the panel might appear too large or too small. The Edit Panel dialog box in the User Interface Editor contains an option to scale the panel to the resolution of the screen. You can adjust resolution from zero to 100 percent. Your value for this option is saved for each panel in the `.uir` file. Scaling of the panel and its contents takes place when you call `LoadPanel` or `LoadPanelEx` in your program.

You can use the `ATTR_RESOLUTION_ADJUSTMENT` system attribute to override the settings in the `.uir` file. To override the setting for a panel, call `SetSystemAttribute` to set `ATTR_RESOLUTION_ADJUSTMENT` before you call `LoadPanel` or `LoadPanelEx`. After calling `LoadPanel` or `LoadPanelEx`, you can call `GetPanelAttribute` with `ATTR_RESOLUTION_ADJUSTMENT` to obtain the setting that was saved in the `.uir` file.

The default value for this attribute is `VAL_USE_PANEL_SETTING`, which you use if you do not want to override the setting in the `.uir` file. Otherwise, pass a value from zero to 100 percent.

The User Interface Library maintains a separate value for this attribute for each thread in your program.

Suppressing Event Processing

If you call `QuitUserInterface` to terminate a `RunUserInterface` call and events already exist 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. The User Interface Library maintains a separate value for this attribute for each thread in your program.

Taskbar Buttons (Windows 95/NT)

The LabWindows/CVI Run-time Engine automatically adds a button to the taskbar for your executable. When you use LabWindows/CVI to create an executable, a taskbar button automatically appears for your executable and contains the text you entered originally in the Application Title field of the Create Standalone Executable dialog box. When you use an external compiler to create executables, this taskbar button contains the filename of the executable by default. Table 3-67 describes how to alter this automatic feature.

Table 3-67. Modifying the Taskbar Button of Your Run-Time Executable

Executable created in CVI?	Calls CVI Libraries?	When does the button appear?	Possible Modifications to the Taskbar Button
yes	optional	Prior to the call to `main` or `WinMain`.	Prevent display of taskbar button using `ATTR_TASKBAR_BUTTON_VISIBLE`. Set the text in the taskbar button through the Application Title field in the Create Standalone Executable dialog box. If you leave the field blank, the filename of the executable appears in the taskbar button. Change the text in the taskbar button using `ATTR_TASKBAR_BUTTON_TEXT`.

no	yes	During the call to `InitCVIRTE`.	Prevent display of taskbar button by inserting the following call before the call to `InitCVIRTE`:
			`SetSystemAttribute (ATTR_TASKBAR_BUTTON_VISIBLE, 0);`
			
			Set the text in the taskbar button by creating a string table resource and adding a string with ID number `65535`.
			
			Alternatively, set the text in the taskbar button by making the following call prior to the call to `InitCVIRTE:`
			`SetSystemAttribute (ATTR_TASKBAR_BUTTON_TEXT, "<text>");`

no	no	Button does not appear because the executable loads CVI libraries indirectly, through a DLL.	Add a button to your taskbar by making the following call from the DLL:
			`SetSystemAttribute (ATTR_TASKBAR_BUTTON_VISIBLE, 1);`
			
			Set the text in the taskbar button by making the following call:
			`SetSystemAttribute (ATTR_TASKBAR_BUTTON_TEXT, "<text>");`

The User Interface Library maintains a separate value for these attributes for each thread in your program.

The attributes mentioned in Table 3-67 have no effect on platforms other than Windows 95/NT.

To create a taskbar button for each panel, use the `ATTR_HAS_TASKBAR_BUTTON` panel attribute.

Generating Hard Copy Output

You can generate hard copy output of panels or individual controls to a graphics printer or a file.

Table 3-68. Functions for Hard Copy Output

Function	Description
`PrintPanel`	Outputs a panel and its child panels.
`PrintCtrl`	Outputs an individual control.
`PrintTextBuffer`	Prints a null-terminated string.
`PrintTextFile`	Prints the contents of an ASCII text file.
`SetPrintAttribute`	Sets hard copy attributes.
`GetPrintAttribute`	Obtains hard copy attributes.

In addition, the following functions can produce hard copy output if you select the hard copy option from the following pop-ups:

```
XGraphPopup
XYGraphPopup
YGraphPopup
WaveformGraphPopup
```

Compatible Printers

Under Windows, LabWindows/CVI can print to any graphics printer with a Windows compatible driver. You select the printer through the Printers utility in the Windows Control Panel or through the LabWindows/CVI print dialog box.

Under UNIX, LabWindows/CVI can print to any graphics printer compatible with the `xpr` printing utility under the X Windows System. You select the printer through the `PRINTER` environment variable, the `printcap` file, or the LabWindows/CVI print dialog box. You set the printer filter in the `.Xdefaults` file.

In addition, you can direct output to a disk file.

Graphics and General Hard Copy Attributes

Table 3-69 contains the list of graphics and general hard copy attributes accessible through `GetPrintAttribute` and `SetPrintAttribute`. All of these attributes affect the behavior of `PrintPanel` and `PrintCtrl`. Some also affect the behavior of `PrintTextBuffer` and `PrintTextFile`.

Table 3-69. Graphics and General Hard Copy Attributes

Attribute	Type	Notes
`ATTR_BITMAP_PRINTING`	integer	`0` = Use direct printing if available, the default under Windows `1` = Use bitmap printing

`ATTR_COLOR_MODE`	integer	`0` = `VAL_BW`
		`1` = `VAL_GRAY_SCALE`
		`2` = `VAL_COLOR`
		Refer to Table 3-72.
		Does not apply to printing text files or buffers.

`ATTR_DUPLEX`	integer	`1` = `VAL_SIMPLEX`, the default
		`2` = `VAL_VERTDUPLEX`
		`3` = `VAL_HORIZDUPLEX`
		`-1` = `VAL_USE_PRINTER_SETTING`

`ATTR_EJECT_AFTER`	integer	`1` = eject page after print `0` = do not eject page after print, the default
`ATTR_NUMCOPIES`	integer	Number of copies; range = 1 to 100. `-1` = `VAL_USE_PRINTER_SETTING`

`ATTR_ORIENTATION`	integer	`1` = `VAL_PORTRAIT`
		`2` = `VAL_LANDSCAPE`, the default
		`-1` = `VAL_USE_PRINTER_SETTING`

`ATTR_PRINT_AREA_HEIGHT`	integer	Millimeter/10 or `-1` = `VAL_USE_ENTIRE_PAPER` `0` = `VAL_INTEGRAL_SCALE`, the default Does not apply to printing text files or buffers.

Table 3-69. Graphics and General Hard Copy Attributes (Continued)

Attribute	Type	Notes
ATTR_PRINT_AREA_WIDTH	integer	Millimeter/10 or -1 = VAL_USE_ENTIRE_PAPER, the default 0 = VAL_INTEGRAL_SCALE Does not apply to printing text files or buffers.
ATTR_PRINTER_NAME	char *	Currently selected printer. If this attribute is NULL, an empty string, or an unknown printer when you call a printing function, the value changes to reference the current system printer.
ATTR_PRINTER_NAME_LENGTH	integer	Number of characters, with the exception of the ASCII NUL byte, in the current value of the ATTR_PRINTER_NAME attribute (GetPrintAttribute only).
ATTR_SYSTEM_PRINT_DIALOG_ONLY	integer	0 = Display the LabWindows/CVI print dialog box. From this dialog box you can open the Windows printer-specific dialog box (the default). 1 = Display only the Windows printer-specific dialog box. Do not show the LabWindows/CVI print dialog box.
ATTR_TEXT_WRAP	integer	Determines whether to wrap text when text extends past the defined width.
ATTR_XOFFSET	integer	Millimeter/10 or VAL_CENTER_ON_PAPER Does not apply to printing text files or buffers.
ATTR_YOFFSET	integer	Millimeter/10 or VAL_CENTER_ON_PAPER Does not apply to printing text files or buffers.
ATTR_XRESOLUTION	integer	Dots Per Inch or VAL_USE_PRINTER_SETTING
ATTR_YRESOLUTION	integer	Dots Per Inch or VAL_USE_PRINTER_SETTING

Text Hard Copy Attributes

Table 3-70 lists hard copy attributes for text that you can access through `GetPrintAttribute` and `SetPrintAttribute`. These attributes affect the behavior of only `PrintTextBuffer` and `PrintTextFile`.

Table 3-70. Hard Copy Attributes for Text Printing

Attribute	Type	Notes
`ATTR_PRINT_BOLD`	integer	Windows only. `1` = text is bold `0` = text is not bold
`ATTR_PRINT_FONT_NAME`	char *	Windows only. Name of the font. Refer to Table 3-5. Default: `"Courier"`
`ATTR_PRINT_FONT_NAME_LENGTH`	integer	Windows only. Number of characters, with the exception of the ASCII NUL byte, in the current value of the `ATTR_PRINT_FONT_NAME` attribute.
`ATTR_PRINT_ITALIC`	integer	Windows only. `1` = text is in italics `0` = text is not in italics
`ATTR_PRINT_POINT_SIZE`	integer	Windows only. Point size of text. Valid range: 6 to 48. Default: `13`
`ATTR_PRINT_STRIKEOUT`	integer	Windows only. `1` = text has strikeout `0` = text does not have strikeout
`ATTR_PRINT_UNDERLINE`	integer	Windows only. `1` = text is underlined `0` = text is not underlined
`ATTR_SHOW_DATE`	integer	`1` = display current date on first line of each page `0` = do not display date
`ATTR_SHOW_FILE_NAME`	integer	`1` = display file name on first line of each page `0` = do not display file name
`ATTR_SHOW_LINE_NUMBERS`	integer	`1` = display line numbers `0` = do not display line numbers

Table 3-70. Hard Copy Attributes for Text Printing (Continued)

Attribute	Type	Notes
`ATTR_SHOW_PAGE_NUMBERS`	integer	`1` = display page numbers `0` = do not display page numbers
`ATTR_SHOW_TIME`	integer	`1` = display current time on first line of each page `0` = do not display time
`ATTR_TAB_INTERVAL`	integer	Number of spaces represented by a <Tab> character. The default is `4`.

Hard Copy Attribute Discussion

The following table contains a more detailed discussion of certain hard copy attributes.

Table 3-71. Detailed Discussion of Certain Hard Copy Attributes

Attribute	Discussion
`ATTR_BITMAP_PRINTING`	When you enable `ATTR_BITMAP_PRINTING`, your printouts reflect the lower resolution of your computer screen, but all objects in your user interface appear correctly in the printout. Generally, you might prefer to leave `ATTR_BITMAP_PRINTING` disabled and rely direct printing, which is the default under Windows. With direct printing, your printouts reflect the superior resolution of the printer. `ATTR_BITMAP_PRINTING` exists for those cases when your printer cannot correctly render certain kinds of objects. For example, in some printers direct printing does not work for transparent bitmaps. Under UNIX, LabWindows/CVI always uses bitmap printing.
`ATTR_DUPLEX`	Determines whether the output is single or double sided. `VAL_VERTDUPLEX` prints double sided with the top side of each page on the opposite edges. `VAL_HORIZDUPLEX` prints double sided with the top side of each page on the same edge.
`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. Because of a limitation in the UNIX `xpr` printing utility, you cannot print multiple outputs on the same page under UNIX when your printer is configured for Postscript.

Table 3-71. Detailed Discussion of Certain Hard Copy Attributes (Continued)

Attribute	Discussion
ATTR_PRINT_AREA_HEIGHT	Sets the vertical dimension of the paper as you view it in portrait orientation, regardless of the ATTR_ORIENTATION setting. VAL_USE_ENTIRE_PAPER uses as much of the paper height as possible. VAL_INTEGRAL_SCALE scales the output to an integral multiple of its size on the screen, to prevent aliasing and distortion.
ATTR_PRINT_AREA_WIDTH	Sets the horizontal dimension of the paper as you view it in portrait orientation, regardless of the ATTR_ORIENTATION setting. VAL_USE_ENTIRE_PAPER uses as much of the paper width as possible. VAL_INTEGRAL_SCALE scales the output to an integral multiple of its size on the screen, to prevent aliasing and distortion.
ATTR_XOFFSET	Sets the *x* offset of the hardcopy image on the paper. The print function measures the *x* offset from the left edge of the paper as you view it in portrait orientation, regardless of the ATTR_ORIENTATION setting. The value 0 forces the image to what is the left edge of the paper in portrait orientation. VAL_CENTER_ON_PAPER centers the image in the x-plane.
ATTR_YOFFSET	Sets the *y* offset of the hardcopy image on the paper. The print function measures the *y* offset from the top edge of the paper as you view it in portrait orientation, regardless of the ATTR_ORIENTATION setting. The value 0 forces the image to what is top edge of the paper in portrait orientation. VAL_CENTER_ON_PAPER centers the image in the y-plane.
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.

Using Printer Settings under Windows

Under Windows, when you set a hardcopy attribute to VAL_USE_PRINTER_SETTING, the next call to a printing function uses the operating system's current setting for the selected printer. After the printing function returns, the value of the attribute is no longer VAL_USE_PRINTER_SETTING. Instead, LabWindows/CVI stores the system's setting as the attribute value. If you want to update the attribute value based on the current system setting each time you print, set the attribute to VAL_USE_PRINTER_SETTING before each call to a printing function.

Hard Copy Color Modes

Table 3-72 shows the values for `ATTR_COLOR_MODE`, depending on the operating system and type of printer.

Table 3-72. Values for ATTR_COLOR_MODE

Type of Printer	Values
default value	`VAL_COLOR`
PC with color printer	`VAL_COLOR`: prints in color
	`VAL_GRAYSCALE`: prints in grayscale
	`VAL_BW`: undefined
PC with non-color printer	`VAL_COLOR`: prints in grayscale
	`VAL_GRAYSCALE`: prints in grayscale
	`VAL_BW`: prints in black and white
Sun with color printer	`VAL_COLOR`: prints in color
	`VAL_GRAYSCALE`: prints in grayscale
	`VAL_BW`: undefined
Sun with non-color printer	`VAL_COLOR`: undefined
	`VAL_GRAYSCALE`: prints in grayscale
	`VAL_BW`: prints in black and white

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: Control callback
- Main callback event: 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 more 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 following events can be swallowed:***

```
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.

The main callback function is the only type of callback function that responds to `EVENT_END_TASK`. This event occurs only under 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. Idle events occur on a regular basis as long as LabWindows/CVI processes events. 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 are called before the main callback so that both have the opportunity to process events.

ProcessDrawEvents

While your program executes in a callback function or in code that does not call `RunUserInterface` or `GetUserEvent`, LabWindows/CVI cannot update the user interface. If a particular function is overly time-consuming, it essentially "locks out" user interface updates. To allow LabWindows/CVI to process these updates, place a call to `ProcessDrawEvents` in your source code.

Note ***LabWindows/CVI automatically updates the user interface in*** `GetUserEvent` ***or when a callback returns.***

ProcessSystemEvents

While your program executes in a callback function or in code that does not call `RunUserInterface` or `GetUserEvent`, LabWindows/CVI does not process user interface and system events. Call `ProcessSystemEvents` to allow LabWindows/CVI to process events. Because `ProcessSystemEvents` can cause other callback functions to execute before it completes, you must use this function with care.

In particular, when `ProcessSystemEvents` handles an event that starts a tracking loop, such as the user pulling down a menu, `ProcessSystemEvents` does not return until the tracking loop completes. In the case of pulling down a menu, the tracking loop does not complete until the user dismisses the menu. Consequently, do not call `ProcessSystemEvents` in your program if you want to continue executing the subsequent lines of code even during a tracking loop.

`ProcessSystemEvents` processes all pending system events, such as,

- Keystrokes, mouse events, and screen updates.
- Events generated by other applications, for example, Windows messages you receive in callbacks you install with `RegisterWinMsgCallback`.

Note ***LabWindows/CVI processes events automatically when you call*** `GetUserEvent` ***or after a callback returns.***

PostDeferredCall

`PostDeferredCall` tells LabWindows/CVI to call a specific function the next time LabWindows/CVI processes events. You typically call `PostDeferredCall` in a function you install as an asynchronous interrupt handler. The types of operations you can perform in an asynchronous interrupt handler are limited. For example, you cannot freely access global variables. Pass to `PostDeferredCall` the name of a function that contains the code you cannot include in the asynchronous interrupt handler.

QueueUserEvent

Use `QueueUserEvent` to place an event in the `GetUserEvent` queue. Event numbers 1,000 to 10,000 are reserved for your use. `GetUserEvent` returns these events, along with commit 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 `RunUserInterface`.

Multithreading in a Windows 95/NT Executable

The User Interface Library is multithread-safe when used under Windows 95/NT in a standalone executable or in a DLL called from a standalone executable. This section discusses the ways that you can use the User Interface Library in a multithreaded program. The section also discusses various behaviors, rules, and restrictions.

Different Approaches to Multithreaded User Interface Programming

You can take three general approaches to the use of multiple threads in conjunction with the User Interface Library.

The first approach is to perform all of your user interface processing in the main thread. You create panels and call functions like `SetCtrlAttribute` and `SetCtrlVal` in the main thread. You use your other threads for I/O processing or data analysis. For example you could acquire data continuously in a separate thread. If you wanted to display the data in the main thread, you could place the acquired data in a global variable and use state variables to indicate how much data was available for display. To protect the data and the state variables from simultaneous access by both threads, you would use the "critical section" facility that the Windows SDK provides.

Another example of the first approach is to create a new thread each time the user clicks on a command button on your user interface panel. Each thread performs the assigned task and then terminates. Because a separate thread performs the user-requested task, the user can continue to operate the user interface panel without waiting for the task to complete.

In the second approach, you create all of your panels in the main thread but call functions such as `SetCtrlAttribute` and `SetCtrlVal` from different threads. For instance, if you are performing data acquisition in a separate thread, you could update a numeric control with a newly acquired data point by calling `SetCtrlVal` from the data acquisition thread.

In the third approach, you create panels in multiple threads. The set of panels in each thread behave almost as if they were in separate processes. For instance, each set of panels is in a separate z-plane grouping. The panels in the thread of the active panel are on top of the panels in the other threads. Also, popup panels are modal only with respect to panels of the same thread. If you use `InstallPopup` to display a dialog box, the end-user cannot access any other panels in the same thread, but is free to operate panels you create in different threads. You can display a separate task bar button for each thread.

Behaviors, Rules, and Restrictions

This section describes how the User Interface Library behaves in a multithreaded program. It also explains various rules and restrictions that you must follow in your program. Most of these rules and restrictions are a consequence of the behavior of the Windows operating system.

Child Threads Should Terminate before Program Ends

You should always make every thread other than the main thread terminate before your program terminates. Otherwise, under Windows 95, the LabWindows/CVI Run-time Engine cannot perform all of its cleanup operations. In addition, if your code is contained within a DLL and an application dynamically loads and unloads the DLL, the application might lose resources that were allocated in the threads you did not terminate.

Panels

You must discard a panel in the same thread in which you load or create it. Before a thread terminates, explicitly discard all panels you loaded or created in that thread.

If you create a child panel, you must do so in the same thread in which you create or load the top-level panel. You can create controls other than timer controls in any thread.

If you call `InstallPopup` on a panel, you must do so in the same thread in which you create or load the panel. The popup is modal only with respect to other panels you load or create in the same thread. You must call `RemovePopup` in the same thread.

You can call `SetPanelPos` and `SetPanelSize` on a top-level panel only in the thread in which you load or create it.

The following panel attributes, when applied to a top-level panel, can be set only in the thread in which you load or create the panel:

`ATTR_FLOATING`

`ATTR_SIZABLE`

`ATTR_MOVABLE`

`ATTR_CAN_MINIMIZE`

`ATTR_CAN_MAXIMIZE`

`ATTR_CLOSE_ITEM_VISIBLE`

`ATTR_SYSTEM_MENU_VISIBLE`

`ATTR_TITLEBAR_VISIBLE`

`ATTR_WIDTH`

`ATTR_HEIGHT`

`ATTR_TOP`

`ATTR_LEFT`

`ATTR_TITLE`

`ATTR_HAS_TASKBAR_BUTTON`

If you enable the `ATTR_FLOATING` attribute on a panel, the panel floats only with respect to panels that you load or create in the same thread.

Event Processing

In general, LabWindows/CVI invokes user interface callbacks in the same thread in which you create the panel or menu bar to which the callback event applies. This applies to all controls on a panel, including timer controls. The following exceptions exist:

- `DiscardCtrl` invokes the control callback with the `EVENT_DISCARD` message. You can call `DiscardCtrl` in any thread.
- For the `EVENT_IDLE` message, LabWindows/CVI invokes the main callback in the thread in which you call `InstallMainCallback`.

Deferred Calls

The `PostDeferredCallToThread` function allows you to schedule a callback function to be called in another thread, called the *target* thread. LabWindows/CVI does not invoke the callback function until it processes events in the target thread.

`PostDeferredCall` always posts the call to the *main* thread. This behavior provides backwards compatibility with LabWindows/CVI programs that make calls to `PostDeferredCall` from asynchronous interrupt callbacks.

Standard I/O Window

Your main thread must process events if you use the Standard Input/Output window. LabWindows/CVI processes events for the Standard Input/Output window in the main thread. If the main thread blocks, calls to functions that access or update the Standard Input/Output window—such as `scanf`, `printf`, `ScanIn`, or `FmtOut`—also block.

Fake Keystroke

`FakeKeystroke` always posts the keystroke event to the thread of the active panel.

Wait Cursor and Mouse Cursor

Each thread maintains the state of the wait cursor independently. Thus, calling `SetWaitCursor` in one thread does *not* cause the wait cursor to appear when a panel you create in another thread is active.

On the other hand, the choice of mouse cursor that appears when the wait cursor is inactive is global across all threads in a process. Thus, calling `SetMouseCursor` changes the mouse cursor regardless of which panel is active.

Sleep Policy

Each thread has a separate sleep policy that you set with `SetSleepPolicy`.

Font Pop-Up Defaults

`SetFontPopupDefaults` maintains separate values for each thread.

Batch Drawing on Canvas Controls

While batch drawing is in effect on a canvas control, do not access the control from any other thread.

File Dialog Boxes

You can display multiple file dialog boxes from different threads at the same time.

Current Working Directory

Do not rely on the concept of a “current working directory” if you perform file I/O or display file dialog boxes from more than one thread. The operating system does *not* maintain a separate working directory for each thread.

Blocking

Generally, each User Interface Library function blocks access to the library from other threads until it returns. Other threads that attempt to call the library must wait until the active call returns. However, the library functions do not block while they wait for events from the operating system, nor do they block when they invoke callback functions.

Normally, the period of time in which the blocking occurs is short. However, some functions, such as graph plotting functions, might take a considerable amount of time to complete, even though they do not process events or invoke callbacks.

Other functions in the User Interface Library do *not* block `PostDeferredCall` and `PostDeferredCallToThread`. Thus, you can call `PostDeferredCall` or `PostDeferredCallToThread` in an asynchronous callback, without a long delay. If calls to these functions occur at the same time in different threads, they might block for a short time to protect the global list of deferred calls.

Windows Messages

`RegisterWinMsgCallback` allows you to register a callback to be invoked when your application receives a Windows message. To send a message to your application, either your application or another application must call the Windows SDK `PostMessage` function. The window handle passed to `PostMessage` determines the thread in which LabWindows/CVI invokes the callback function.

To send a message to a particular thread, pass the window handle that `GetCVIWindowHandle` returns when you call it from that thread.

Printing

When you use `PrintPanel`, `PrintCtrl`, `PrintTextBuffer`, or `PrintTextFile` to print a thread, these functions block other threads that attempt to print until the active print function returns. This blocking occurs even when the print function displays a dialog box and waits for user events.

If you set `ATTR_EJECT_ATTR` to zero in one thread, the paper is not ejected even when the next call to a print function comes from another thread. For this reason, use caution when you print from multiple threads.

System Attributes

The following table indicates, for each system attribute, whether the User Interface Library maintains separate values for each thread or one global value for the entire process.

System Attribute	Values Maintained
`ATTR_RESOLUTION_ADJUSTMENT`	Separate value for each thread.
`ATTR_TASKBAR_BUTTON_VISIBLE`	Separate value for each thread.
`ATTR_TASKBAR_BUTTON_TEXT`	Separate value for each thread.
`ATTR_REPORT_LOAD_FAILURE`	One global value.
`ATTR_ALLOW_MISSING_CALLBACKS`	One global value.
`ATTR_ALLOW_UNSAFE_TIMER_EVENTS`	One global value.
`ATTR_SUPPRESS_EVENT_PROCESSING`	Separate value for each thread.

Timer Controls and Events

You can create or discard a timer control only in the thread in which you load or create the panel that contains it.

`ResumeTimerCallbacks` and `SuspendTimerCallbacks` affect only timer controls loaded or created in the active thread.

When you call `ResetTimer` with the first parameter set to `-1`, it resets all timer controls loaded or created in the active thread.

LabWindows/CVI invokes timer callbacks only in the thread in which you created the timer control. Consequently, making a program multithreaded can *reduce* the accuracy of the intervals between one timer callback invocation and another.

For the most accurate timer possible, use the AsyncTmr instrument driver. The AsyncTmr instrument driver uses the Windows multimedia timer. On each occurrence of a timer event, it creates a high-priority thread and invokes a callback. You can use the AsyncTmr instrument driver even in the LabWindows/CVI development environment. Use the following path to locate the AsyncTmr instrument driver:

```
<cvi>\toolslib\toolbox\asynctmr.*
```

4 User Interface Library Reference

This chapter describes the functions in the LabWindows/CVI 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 following section describes the User Interface Library function tree.

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.
- **Mouse and Cursor** is a class of functions that get and set the state of the mouse, the mouse cursor, and the wait cursor.
- **Rectangles and Points** is a class of functions that create and manipulate `Rect` and `Point` structures, which you use to specify locations and areas in Cartesian coordinates systems on canvas controls and bitmaps.
- **Bitmaps** is a class of functions that create and discard bitmaps, which are two-dimensional grids of pixels representing an image.
- **Clipboard** is a class of functions that access the system clipboard.
- **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.

The User Interface Library Function Tree

The following list shows the structure of the User Interface Library function tree:

- Panels
- Menu Structures
  - Menu Bars
  - Menus
  - Menu Items
- Controls/Graphs/Strip Charts
  - General Functions
  - List (Label/Value) Controls
  - Text Boxes
  - Graphs and Strip Charts
    - Graph Plotting and Deleting
    - Graph Cursors
    - Strip Chart Traces
    - Axis Scaling
    - Axis Label Strings
  - Pictures
  - Canvas
    - Drawing
    - Batch Drawing
    - Pens
    - Clipping
    - Accessing Pixel Values
    - Miscellaneous
  - Timers
- Pop-up Panels

- Callback Functions
    - Windows Interrupt Support
- User Interface Management
- Printing
- Mouse and Cursor
- Rectangles and Points
    - Creating and Modifying
    - Retrieving and Comparing Values
- Bitmaps
- Clipboard
- Miscellaneous
- LW DOS Compatibility Functions
- Get Error String

Table 4-1 lists the User Interface Library class and panel names, along with corresponding function names in monospaced text.

Table 4-1. Functions in the User Interface Library Reference Function Tree

Class/Panel Name	**Function Name**
Panels	
Load Panel	`LoadPanel`
Load Panel, Extended	`LoadPanelEx`
New Panel	`NewPanel`
Discard Panel	`DiscardPanel`
Duplicate Panel	`DuplicatePanel`
Display Panel	`DisplayPanel`
Hide Panel	`HidePanel`
Get Active Panel	`GetActivePanel`
Set Active Panel	`SetActivePanel`
Validate Panel	`ValidatePanel`
Default Panel	`DefaultPanel`
Save Panel State	`SavePanelState`
Recall Panel State	`RecallPanelState`
Get Panel Attribute	`GetPanelAttribute`
Set Panel Attribute	`SetPanelAttribute`
Set Panel Size	`SetPanelSize`
Set Panel Position	`SetPanelPos`
Menu Structures	
Menu Bars	
Load Menu Bar	`LoadMenuBar`
Load Menu Bar, Extended	`LoadMenuBarEx`
New Menu Bar	`NewMenuBar`
Discard Menu Bar	`DiscardMenuBar`
Set Panel Menu Bar	`SetPanelMenuBar`
Get Panel Menu Bar	`GetPanelMenuBar`
Get Menu Bar Attribute	`GetMenuBarAttribute`
Set Menu Bar Attribute	`SetMenuBarAttribute`

Table 4-1. Functions in the User Interface Library Reference Function Tree (Continued)

Class/Panel Name	Function Name
Menu Structures (continued)	
Menu Bars (continued)	
Empty Menu Bar	`EmptyMenuBar`
Get Shared Menu Bar Event Panel	`GetSharedMenuBarEventPanel`
Menus	
New Menu	`NewMenu`
Discard Menu	`DiscardMenu`
Empty Menu	`EmptyMenu`
New SubMenu	`NewSubMenu`
Discard SubMenu	`DiscardSubMenu`
Run Popup Menu	`RunPopupMenu`
Menu Items	
New Menu Item	`NewMenuItem`
Discard Menu Item	`DiscardMenuItem`
Insert Separator	`InsertSeparator`
Controls/Graphs/Strip Charts	
General Functions	
New Control	`NewCtrl`
Duplicate Control	`DuplicateCtrl`
Discard Control	`DiscardCtrl`
Get Active Control	`GetActiveCtrl`
Set Active Control	`SetActiveCtrl`
Default Control Value	`DefaultCtrl`
Get Control Value	`GetCtrlVal`
Set Control Value	`SetCtrlVal`
Get Control Attribute	`GetCtrlAttribute`
Set Control Attribute	`SetCtrlAttribute`
Get Control Bounding Rectangle	`GetCtrlBoundingRect`
List (Label/Value) Controls	
Insert List Item	`InsertListItem`
Replace List Item	`ReplaceListItem`
Delete List Item	`DeleteListItem`
Get Value From Index	`GetValueFromIndex`
Get Value Length From Index	`GetValueLengthFromIndex`
Get Index From Value	`GetIndexFromValue`
Get Control Index	`GetCtrlIndex`
Set Control Index	`SetCtrlIndex`
Clear List Control	`ClearListCtrl`
Get Number of List Items	`GetNumListItems`
Get List Item Image	`GetListItemImage`
Set List Item Image	`SetListItemImage`
Get Label From Index	`GetLabelFromIndex`
Get Label Length From Index	`GetLabelLengthFromIndex`
Is List Item Checked	`IsListItemChecked`

Table 4-1. Functions in the User Interface Library Reference Function Tree (Continued)

Class/Panel Name	Function Name
Controls/Graphs/Strip Charts (continued)	
List (Label/Value) Controls (continued)	
Check List Item	`CheckListItem`
Get Number of Checked Items	`GetNumCheckedItems`
Text Boxes	
Insert Text Box Line	`InsertTextBoxLine`
Replace Text Box Line	`ReplaceTextBoxLine`
Delete Text Box Line	`DeleteTextBoxLine`
Get Number of Text Box Lines	`GetNumTextBoxLines`
Reset Text Box	`ResetTextBox`
Get Text Box Line	`GetTextBoxLine`
Get Text Box Line Length	`GetTextBoxLineLength`
Get Text Box Line Offset	`GetTextBoxLineOffset`
Get Text Box Line From Offset	`GetTextBoxLineIndexFromOffset`
Graphs and Strip Charts	
Graph Plotting and Deleting	
Plot X	`PlotX`
Plot Y	`PlotY`
Plot X-Y	`PlotXY`
Plot Waveform	`PlotWaveform`
Plot Point	`PlotPoint`
Plot Text	`PlotText`
Plot Line	`PlotLine`
Plot Rectangle	`PlotRectangle`
Plot Polygon	`PlotPolygon`
Plot Oval	`PlotOval`
Plot Arc	`PlotArc`
Plot Intensity	`PlotIntensity`
Plot Scaled Intensity	`PlotScaledIntensity`
Plot Bitmap	`PlotBitmap`
Delete Graph Plot	`DeleteGraphPlot`
Get Plot Attribute	`GetPlotAttribute`
Set Plot Attribute	`SetPlotAttribute`
Refresh Graph	`RefreshGraph`
Graph Cursors	
Get Graph Cursor	`GetGraphCursor`
Set Graph Cursor	`SetGraphCursor`
Get Active Graph Cursor	`GetActiveGraphCursor`
Set Active Graph Cursor	`SetActiveGraphCursor`
Get Graph Cursor Index	`GetGraphCursorIndex`
Set Graph Cursor Index	`SetGraphCursorIndex`
Get Cursor Attribute	`GetCursorAttribute`
Set Cursor Attribute	`SetCursorAttribute`

Table 4-1. Functions in the User Interface Library Reference Function Tree (Continued)

Class/Panel Name	**Function Name**
Controls/Graphs/Strip Charts (continued)	
Graphs and Strip Charts (continued)	
Strip Chart Traces	
Plot Strip Chart	`PlotStripChart`
Plot Strip Chart Point	`PlotStripChartPoint`
Clear Strip Chart	`ClearStripChart`
Get Trace Attribute	`GetTraceAttribute`
Set Trace Attribute	`SetTraceAttribute`
Axis Scaling	
Get Axis Scaling Mode	`GetAxisScalingMode`
Set Axis Scaling Mode	`SetAxisScalingMode`
Get Axis Range (Obsolete)	`GetAxisRange`
Set Axis Range (Obsolete)	`SetAxisRange`
Axis Label Strings	
Insert Axis Item	`InsertAxisItem`
Replace Axis Item	`ReplaceAxisItem`
Delete Axis Item	`DeleteAxisItem`
Clear Axis Items	`ClearAxisItems`
Get Number of Axis Items	`GetNumAxisItems`
Get Axis Item Label and Value	`GetAxisItem`
Get Axis Item Label Length	`GetAxisItemLabelLength`
Pictures	
Display Image File	`DisplayImageFile`
Delete Image	`DeleteImage`
Get Image Info (Obsolete)	`GetImageInfo`
Get Image Bits (Obsolete)	`GetImageBits`
Set Image Bits (Obsolete)	`SetImageBits`
Alloc Image Bits (Obsolete)	`AllocImageBits`
Canvas	
Drawing	
Draw Point	`CanvasDrawPoint`
Draw Line	`CanvasDrawLine`
Draw Line To	`CanvasDrawLineTo`
Draw Rectangle	`CanvasDrawRect`
Dim Rectangle	`CanvasDimRect`
Draw Rounded Rectangle	`CanvasDrawRoundedRect`
Draw Oval	`CanvasDrawOval`
Draw Arc	`CanvasDrawArc`
Draw Poly	`CanvasDrawPoly`
Draw Text in Rectangle	`CanvasDrawText`
Draw Text at Point	`CanvasDrawTextAtPoint`
Draw Bitmap	`CanvasDrawBitmap`
Scroll	`CanvasScroll`
Invert Rectangle	`CanvasInvertRect`
Clear	`CanvasClear`

Table 4-1. Functions in the User Interface Library Reference Function Tree (Continued)

Class/Panel Name	**Function Name**
Controls/Graphs/Strip Charts (continued)	
Canvas (continued)	
Batch Drawing	
Start Batch Drawing	`CanvasStartBatchDraw`
End Batch Drawing	`CanvasEndBatchDraw`
Pens	
Set Pen Position	`CanvasSetPenPosition`
Get Pen Position	`CanvasGetPenPosition`
Set Pen Attributes To Defaults	`CanvasDefaultPen`
Clipping	
Set Clipping Rectangle	`CanvasSetClipRect`
Get Clipping Rectangle	`CanvasGetClipRect`
Accessing Pixel Values	
Get a Single Pixel Value	`CanvasGetPixel`
Get Pixel Values	`CanvasGetPixels`
Miscellaneous	
Update Canvas	`CanvasUpdate`
Timers	
Reset Timer	`ResetTimer`
Suspend Timer Callbacks	`SuspendTimerCallbacks`
Resume Timer Callbacks	`ResumeTimerCallbacks`
Pop-up Panels	
Install Popup	`InstallPopup`
Remove Popup	`RemovePopup`
Message Popup	`MessagePopup`
Confirm Popup	`ConfirmPopup`
Prompt Popup	`PromptPopup`
Generic Message	`GenericMessagePopup`
File Select Popup	`FileSelectPopup`
Multifile Select Popup	`MultiFileSelectPopup`
Directory Select Popup	`DirSelectPopup`
X Graph Popup	`XGraphPopup`
Y Graph Popup	`YGraphPopup`
X-Y Graph Popup	`XYGraphPopup`
Waveform Graph Popup	`WaveformGraphPopup`
Get System Popups Attribute	`GetSystemPopupsAttribute`
Set System Popups Attribute	`SetSystemPopupsAttribute`
Font Select Popup	`FontSelectPopup`
Set Font Select Popup Defaults	`SetFontPopupDefaults`
Callback Functions	
Install Main Callback	`InstallMainCallback`
Install Control Callback	`InstallCtrlCallback`
Install Panel Callback	`InstallPanelCallback`
Install Menu Callback	`InstallMenuCallback`
Install Menu Dimmer Callback	`InstallMenuDimmerCallback`

Table 4-1. Functions in the User Interface Library Reference Function Tree (Continued)

Class/Panel Name	**Function Name**
Callback Functions (continued)	
Post Deferred Call	`PostDeferredCall`
Post Deferred Call to Thread	`PostDeferredCallToThread`
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	`RunUserInterface`
Quit User Interface	`QuitUserInterface`
Get User Event	`GetUserEvent`
Set Input Mode	`SetInputMode`
Process Draw Events	`ProcessDrawEvents`
Process System Events	`ProcessSystemEvents`
Queue User Event	`QueueUserEvent`
Set Idle Event Rate	`SetIdleEventRate`
Fake Keystroke	`FakeKeystroke`
Get Sleep Policy	`GetSleepPolicy`
Set Sleep Policy	`SetSleepPolicy`
Printing	
Get Print Attribute	`GetPrintAttribute`
Set Print Attribute	`SetPrintAttribute`
Print Control	`PrintCtrl`
Print Panel	`PrintPanel`
Print Text File	`PrintTextFile`
Print Text Buffer	`PrintTextBuffer`
Mouse and Cursor	
Get Wait Cursor	`GetWaitCursorState`
Set Wait Cursor	`SetWaitCursor`
Get Mouse Cursor	`GetMouseCursor`
Set Mouse Cursor	`SetMouseCursor`
Get Global Mouse State	`GetGlobalMouseState`
Get Relative Mouse State	`GetRelativeMouseState`
Rectangles and Points	
Creating and Modifying	
Make Rect	`MakeRect`
Set Rect Coordinates	`RectSet`
Set Rect Coords From Points	`RectSetFromPoints`
Set Bottom Edge of Rect	`RectSetBottom`
Set Right Edge of Rect	`RectSetRight`
Set Center Point of Rect	`RectSetCenter`
Offset Rect	`RectOffset`
Move Rect	`RectMove`
Grow (or Shrink) Rect	`RectGrow`

Table 4-1. Functions in the User Interface Library Reference Function Tree (Continued)

Class/Panel Name	**Function Name**
Rectangles and Points (continued)	
Creating and Modifying (continued)	
Make Point	`MakePoint`
Set Point Coordinates	`PointSet`
Retrieving and Comparing Values	
Get Rect Bottom	`RectBottom`
Get Rect Right	`RectRight`
Get Rect Center	`RectCenter`
Are Rects Equal?	`RectEqual`
Is Rect Empty?	`RectEmpty`
Does Rect Contain Point?	`RectContainsPoint`
Does Rect Contain Rect?	`RectContainsRect`
Are Rects the Same Size?	`RectSameSize`
Calculate Rect Union	`RectUnion`
Calculate Rect Intersection	`RectIntersection`
Are Points Equal?	`PointEqual`
Calculate Point Pinned to Rect	`PointPinnedToRect`
Bitmaps	
Create New Bitmap	`NewBitmap`
Get Bitmap From a File	`GetBitmapFromFile`
Get Bitmap From a Control	`GetCtrlBitmap`
Get Control Display Bitmap	`GetCtrlDisplayBitmap`
Get Scaled Control Bitmap	`GetScaledCtrlDisplayBitmap`
Get Panel Display Bitmap	`GetPanelDisplayBitmap`
Get Scaled Panel Bitmap	`GetScaledPanelDisplayBitmap`
Get Bitmap Info	`GetBitmapInfo`
Get Bitmap Data	`GetBitmapData`
Alloc Bitmap Data	`AllocBitmapData`
Set Bitmap Data	`SetBitmapData`
Set Control Bitmap	`SetCtrlBitmap`
Discard Bitmap	`DiscardBitmap`
Clipboard	
Get Text From Clipboard	`ClipboardGetText`
Put Text On Clipboard	`ClipboardPutText`
Get Bitmap From Clipboard	`ClipboardGetBitmap`
Put Bitmap on Clipboard	`ClipboardPutBitmap`
Miscellaneous	
Make Color	`MakeColor`
Get 3d Border Colors	`Get3dBorderColors`
Create Meta Font	`CreateMetaFont`
Get Text Display Size	`GetTextDisplaySize`
Get Screen Size	`GetScreenSize`
Get System Attribute	`GetSystemAttribute`
Set System Attribute	`SetSystemAttribute`

Table 4-1. Functions in the User Interface Library Reference Function Tree (Continued)

Class/Panel Name	Function Name
Miscellaneous (continued)	
Make Application Active	`MakeApplicationActive`
Minimize All Windows	`MinimizeAllWindows`
LW DOS Compatibility Functions	
Configure Printer	`ConfigurePrinter`
Display PCX File	`DisplayPCXFile`
DOS Color to RGB	`DOSColorToRGB`
DOS Compatibility Window	`DOSCompatWindow`
Get Error String	
Get Error String	`GetUILErrorString`

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 `GetUILErrorString` 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 appear in alphabetical order.

AllocBitmapData

```
int status = AllocBitmapData (int bitmapID, int **colorTable, char **bits,
                              unsigned char **mask);
```

Purpose

Allocates the buffers you pass to `GetBitmapData`. If you use `GetBitmapInfo`, you must allocate the buffers yourself.

You must free the buffers when you are done with them.

Parameters

Input

Name	Type	Description
bitmapID	integer	ID of the bitmap object containing the image. You obtain the ID from `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`.

Output

Name	Type	Description
colorTable	pointer to integer	Pointer variable in which to store the address of the allocated color table buffer.
bits	pointer to unsigned char	Pointer variable in which to store the address of the allocated bits data buffer.
mask	pointer to unsigned char	Pointer variable in which to store the address of the allocated mask buffer.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You can pass `NULL` for any of the **colorTable**, **bits**, or **mask** parameters if you do not want `AllocBitmapData` to allocate the corresponding buffer.

If the image does not exist, `AllocBitmapData` sets the **colorTable**, **bits**, and **mask** parameters to `NULL`. It sets the **colorTable** parameter to `NULL` if the pixel depth of the image is greater than eight. It sets the **mask** parameter to `NULL` if the image does not have a mask.

Caution ***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

```
int status = AllocImageBits(int panelHandle, int controlID, int imageID,
                            int **colorTable, unsigned char **bitmap,
                            unsigned char **mask);
```

Purpose

Allocates the buffers you pass to `GetImageBits`. `AllocImageBits` 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 when you are done with them.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
imageID	integer	For a picture ring, the zero-based index of an image in the ring. For a graph, the **plotHandle** you obtain from `PlotBitmap`. For picture controls and buttons, this parameter is ignored.

Output

Name	Type	Description
colorTable	pointer to integer	Pointer variable in which to store the address of the allocated color table buffer.
bitmap	pointer to unsigned char	Pointer variable in which to store the address of the allocated bitmap buffer.
mask	pointer to unsigned char	Pointer variable in which to store the address of the allocated mask buffer.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You can pass `NULL` for any of the **colorTable**, **bitmap**, and **mask** parameters if you do not want `AllocImageBits` to allocate the corresponding buffer.

If the image does not exist, `AllocImageBits` sets the **colorTable**, **bitmap**, and **mask** parameters to `NULL`. It sets the **mask** parameter to `NULL` if the image does not have a mask.

Caution *You must free the* **colorTable, bitmap,** *and* **mask** *buffers when you are done with them. Use the ANSI C Library* `free` *function.*

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. You can set the background color of the canvas control through the `ATTR_PICT_BGCOLOR` attribute.

Unlike other canvas drawing operations, `CanvasClear` can clear the canvas control beyond the canvas clipping rectangle.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
rect	Rect	`Rect` structure specifying the location and size of the rectangle to clear. Use `VAL_ENTIRE_OBJECT` to specify the entire canvas.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakeRect`

CanvasDefaultPen

```
int status = CanvasDefaultPen (int panelHandle, int controlID);
```

Purpose

Sets all of the attributes of the canvas pen to the default values. The defaults appear in Table 4-2.

Table 4-2. Canvas Pen Default Values

Canvas Pen Attribute	Default Value
`ATTR_PEN_WIDTH`	`1`
`ATTR_PEN_STYLE`	`VAL_SOLID`
`ATTR_PEN_COLOR`	`VAL_BLACK`
`ATTR_PEN_FILL_COLOR`	`VAL_BLACK`
`ATTR_PEN_MODE`	`VAL_COPY_MODE`
`ATTR_PEN_PATTERN`	A solid pattern, expressed as an array of eight unsigned characters, each of which is 0xFF.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`CanvasSetPenPosition`, `CanvasGetPenPosition`, `CanvasDrawLineTo`

CanvasDimRect

```
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.

`CanvasDimRect` draws the checkerboard pattern using the current value of the following attribute: `ATTR_PEN_FILL_COLOR`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
rect	Rect	`Rect` structure specifying the location and size of the area to be dimmed. Use `VAL_ENTIRE_OBJECT` to specify the entire canvas.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakeRect`

CanvasDrawArc

```
int status = CanvasDrawArc (int panelHandle, int controlID, Rect rect,
                            int drawMode, int beginningAngle, int arcAngle);
```

Purpose

Draws an arc on the canvas control. You define the arc 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, counter-clockwise, up to 3,600, to extend the arc.

`CanvasDrawArc` draws the frame of the arc using the current value of the following attributes:

`ATTR_PEN_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_WIDTH`
`ATTR_PEN_STYLE` (ignored under Windows when pen width is greater than one)

`CanvasDrawArc` draws interior of the arc 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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Name	Type	Description
rect	Rect	`Rect` structure specifying the location and size of the rectangle within which to draw the arc.

drawMode	integer	Specifies whether to draw the arc frame, or interior, or both. Valid values:
		`VAL_DRAW_FRAME`
		`VAL_DRAW_INTERIOR`
		`VAL_DRAW_FRAME_AND_INTERIOR`

beginningAngle	integer	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.
arcAngle	integer	How far around the oval, counter-clockwise, up to 3,600, to extend the arc. Specified in tenths of degrees. Negative values are valid.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakeRect, CanvasDrawOval`

CanvasDrawBitmap

```
int status = CanvasDrawBitmap (int panelHandle, int controlID, int bitmapID,
                     Rect sourceRect, Rect destinationRect);
```

Purpose

Draws a bitmap image or portion thereof in the destination rectangle you specify on a canvas control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
bitmapID	integer	ID of the bitmap object containing the image. You obtain the ID from `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`.
sourceRect	Rect	`Rect` structure specifying the portion of the bitmap to draw. 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.
destinationRect	Rect	`Rect` structure specifying the size and location of the area in which to draw the bitmap image on the canvas control. If **sourceRect** and **destinationRect** are not the same size, the bitmap stretches or shrinks to fit.

Return Value

Name	Type	Description
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, 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:

```
CanvasDrawBitmap (panelHandle, controlID, bitmapID, VAL_ENTIRE_OBJECT,
                MakeRect(20,30, VAL_KEEP_SAME_SIZE, VAL_KEEP_SAME_SIZE));
```

See Also

`MakeRect`

CanvasDrawLine

```
int status = CanvasDrawLine (int panelHandle, int controlID, Point start,
                             Point end);
```

Purpose

Draws a line between two specified points.

`CanvasDrawLine` draws the line using the current value of the following attributes:

`ATTR_PEN_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_WIDTH`
`ATTR_PEN_STYLE` (ignored under Windows when pen width is greater than one)

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
start	Point	`Point` structure specifying the location at which the line begins.
end	Point	`Point` structure specifying the location at which the line ends.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakePoint`, `CanvasDrawLineTo`

CanvasDrawLineTo

```
int status = CanvasDrawLineTo (int panelHandle, int controlID, Point end);
```

Purpose

Draws a line between the current pen position and an end point you specify, and sets the pen position to the end point.

`CanvasDrawLineTo` draws the line using the current value of the following attributes:

`ATTR_PEN_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_WIDTH`
`ATTR_PEN_STYLE` (ignored under Windows when pen width is greater than one)

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
end	Point	`Point` structure specifying the location at which the line ends.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakePoint`, `CanvasGetPenPosition`, `CanvasSetPenPosition`, `CanvasDefaultPen`, `CanvasDrawLine`

CanvasDrawOval

```
int status = CanvasDrawOval (int panelHandle, int controlID, Rect rect,
                             int drawMode);
```

Purpose

Draws an oval on the canvas control within the specified rectangle.

`CanvasDrawOval` draws the frame of the oval using the current value of the following attributes:

`ATTR_PEN_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_WIDTH`
`ATTR_PEN_STYLE` (ignored under Windows when pen width is greater than one)

`CanvasDrawOval` draws the interior of the oval using the current value of the following attributes:

`ATTR_PEN_FILL_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_PATTERN`

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
rect	Rect	`Rect` structure specifying the location and size of the rectangle within which to draw the oval.

drawMode	integer	Specifies whether to draw the oval frame, or interior, or both. Valid values:
		`VAL_DRAW_FRAME`
		`VAL_DRAW_INTERIOR`
		`VAL_DRAW_FRAME_AND_INTERIOR`

Return Value

Name	Type	Description
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 at a position you specify.

`CanvasDrawPoint` draws the point using the current value of the following attributes:

```
ATTR_PEN_COLOR
ATTR_PEN_MODE
ATTR_PEN_WIDTH
```

At pen widths of greater than one, the point might appear to be non-circular.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
point	Point	`Point` structure specifying the location at which to draw the point.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakeRect`, `CanvasDrawArc`

CanvasDrawPoly

```
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.

`CanvasDrawPoly` draws the frame of the polygon using the current value of the following attributes:

`ATTR_PEN_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_WIDTH`
`ATTR_PEN_STYLE` (ignored under Windows when pen width is greater than one)

`CanvasDrawPoly` draws the interior of the polygon using the current value of the following attributes:

`ATTR_PEN_FILL_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_PATTERN`

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
numberOfPoints	integer	Number of vertices in the polygon.
points	Point array	Array of Point structures specifying the locations of the vertices of the polygon.
wrap	integer	If non-zero, the function draws a line between the last point and the first point, thereby closing the polygon frame. The function ignores the value when drawing only the interior.

drawMode	integer	Specifies whether to draw the polygon frame, or interior, or both. Valid values:
		`VAL_DRAW_FRAME`
		`VAL_DRAW_INTERIOR`
		`VAL_DRAW_FRAME_AND_INTERIOR`

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

CanvasDrawRect

```
int status = CanvasDrawRect (int panelHandle, int controlID, Rect rect,
                             int drawMode);
```

Purpose

Draws a rectangle on the canvas control.

`CanvasDrawRect` draws the frame of the rectangle using the current value of the following attributes:

`ATTR_PEN_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_WIDTH`
`ATTR_PEN_STYLE` (ignored under Windows when pen width is greater than one)

`CanvasDrawRect` draws the interior of the rectangle using the current value of the following attributes:

`ATTR_PEN_FILL_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_PATTERN`

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
rect	Rect	`Rect` structure specifying the location and size of the rectangle to be drawn.

drawMode	integer	Specifies whether to draw the rectangle frame, or interior, or both. Valid values:
		`VAL_DRAW_FRAME`
		`VAL_DRAW_INTERIOR`
		`VAL_DRAW_FRAME_AND_INTERIOR`

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakeRect`, `CanvasDrawRoundedRect`

CanvasDrawRoundedRect

```
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.

`CanvasDrawRoundedRect` draws the frame of the rectangle using the current value of the following attributes:

`ATTR_PEN_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_WIDTH`
`ATTR_PEN_STYLE` (ignored under Windows when pen width is greater than one)

`CanvasDrawRoundedRect` draws the interior of the rectangle using the current value of the following attributes:

`ATTR_PEN_FILL_COLOR`
`ATTR_PEN_MODE`
`ATTR_PEN_PATTERN`

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
rect	Rect	`Rect` structure specifying the location and size of the rectangle to be drawn.
ovalHeight	integer	Vertical diameter of the oval, the quadrants of which the function draws at the corners of the rounded rectangle.
ovalWidth	integer	Horizontal diameter of the oval, the quadrants of which the function draws at the corners of the rounded rectangle.

drawMode	integer	Specifies whether to draw the rectangle frame, or interior, or both. Valid values:
		`VAL_DRAW_FRAME`
		`VAL_DRAW_INTERIOR`
		`VAL_DRAW_FRAME_AND_INTERIOR`

Return Value

Name	Type	Description
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, `CanvasDrawText` clips it.

`CanvasDrawText` draws the text using the current value of the following attribute: `ATTR_PEN_COLOR`.

`CanvasDrawText` draws the background rectangle using the current value of the following attributes:

```
ATTR_PEN_FILL_COLOR
ATTR_PEN_MODE
ATTR_PEN_PATTERN
```

If you do not want to draw the background of the rectangle, set the `ATTR_PEN_FILL_COLOR` attribute of the canvas control to `VAL_TRANSPARENT`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
text	string	Text string to draw within the rectangle.
metaFont	string	Specifies the text font. Must be one of the predefined metafonts (refer to Table 3-5 in this manual) or a metafont you create by calling `CreateMetaFont`.

Name	Type	Description
bounds	Rect	`Rect` structure specifying location and size of the background rectangle within which to draw the text.
alignment	integer	Determines the placement of the text string within the background rectangle. Refer to the following discussion.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

Specify values in the **bounds** parameter in terms of pixel coordinates, with the origin (0,0) at the upper left corner of the canvas control.

If you want the function to adjust the size of the background rectangle 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** parameter are listed in Table 4-3.

Table 4-3. Alignment Parameter Valid Values

Value	**Description**
`VAL_LOWER_LEFT`	Draw the string in the lower left corner of the background rectangle.
`VAL_CENTER_LEFT`	Start the string from the midpoint of the left edge of the background rectangle.
`VAL_UPPER_LEFT`	Draw the string in the upper left corner of the background rectangle.
`VAL_LOWER_CENTER`	Center the string just above the bottom edge of the background rectangle.
`VAL_CENTER_CENTER`	Center the string in the middle of the background rectangle.
`VAL_UPPER_CENTER`	Center the string just below the top edge of the background rectangle.
`VAL_LOWER_RIGHT`	Draw the string in the lower right corner of the background rectangle.
`VAL_CENTER_RIGHT`	Draw the string so that it ends just at the midpoint of the right edge of the background rectangle.
`VAL_UPPER_RIGHT`	Draw the string in the upper right corner of the background rectangle.

If the background rectangle you specify in **bounds** is smaller than the text display size, the function clips the text to the rectangle and ignores the alignment parameter. If the rectangle width is smaller than the text display width, the text starts from the left. If the rectangle height is smaller than the text display height, the text starts from the top.

See Also

`MakeRect`, `CanvasDrawTextAtPoint`

CanvasDrawTextAtPoint

```
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. If the string exceeds the size of the rectangle, `CanvasDrawTextAtPoint` clips it.

`CanvasDrawTextAtPoint` draws the text using the current value of the following attribute: `ATTR_PEN_COLOR`.

`CanvasDrawTextAtPoint` draws the background of the text using the current value of the following attributes:

```
ATTR_PEN_FILL_COLOR
ATTR_PEN_MODE
ATTR_PEN_PATTERN
```

If you do not want to draw the background of the rectangle, set the `ATTR_PEN_FILL_COLOR` attribute of the canvas control to `VAL_TRANSPARENT`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
text	string	Text string to be drawn at the anchor point.
metaFont	string	Specifies the text font. Must be one of the predefined metafonts (refer to Table 3-5 in this manual) or a metafont you create by calling `CreateMetaFont`.

Name	Type	Description
anchorPoint	Point	`Point` structure specifying location of the point at which to draw the text.
alignment	integer	Determines the placement of the text string in relation to the anchor point. Refer to discussion below.

Return Value

Name	Type	Description
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 you specify with the **alignment** parameter is at the location you specify with the **anchorPoint** parameter. The valid values for the **alignment** parameter are listed in Table 4-4.

Table 4-4. Alignment Parameter Valid Values

Value	Description
`VAL_LOWER_LEFT`	Draw the string so that lower left corner of its enclosing rectangle is at the location specified by **anchorPoint**.
`VAL_CENTER_LEFT`	Draw the string so that midpoint of the left edge of its enclosing rectangle is at the location specified by **anchorPoint**.
`VAL_UPPER_LEFT`	Draw the string so that upper left corner of its enclosing rectangle is at the location specified by **anchorPoint**.
`VAL_LOWER_CENTER`	Draw the string so that midpoint of the bottom edge of its enclosing rectangle is at the location specified by **anchorPoint**.
`VAL_CENTER_CENTER`	Draw the string so that center of its enclosing rectangle is at the location specified by **anchorPoint**.
`VAL_UPPER_CENTER`	Draw the string so that midpoint of the top edge of its enclosing rectangle is at the location specified by **anchorPoint**.
`VAL_LOWER_RIGHT`	Draw the string so that lower right corner of its enclosing rectangle is at the location specified by **anchorPoint**.
`VAL_CENTER_RIGHT`	Draw the string so that midpoint of the right edge of its enclosing rectangle is at the location specified by **anchorPoint**.
`VAL_UPPER_RIGHT`	Draw the string so that upper right corner of its enclosing rectangle is at the location specified by **anchorPoint**.

See Also

`MakeRect`, `CanvasDrawText`

CanvasEndBatchDraw

```
int nestingDepth = CanvasEndBatchDraw (int panelHandle, int controlID);
```

Purpose

Ends the batch drawing you start with `CanvasStartBatchDraw`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
nestingDepth	integer	Number of calls to `CanvasStartBatchDraw` that have not been matched by calls to `CanvasEndBatchDraw` (including this call). A negative value indicates that an error occurred. Refer to Appendix A for error codes.

See Also

`CanvasStartBatchDraw`

CanvasGetClipRect

```
int status = CanvasGetClipRect (int panelHandle, int controlID,
                                Rect *clipRect);
```

Purpose

Obtains the current clipping rectangle for the canvas control. Canvas drawing operations do not extend beyond the clipping rectangle. Any drawing outside the clipping rectangle has no effect. Exception: `CanvasClear` is not limited to the clipping rectangle.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
clipRect	Rect	`Rect` structure in which to store the location and size of the clipping rectangle. If clipping is disabled (the default state), the function sets the height and width values in the structure to zero.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`CanvasSetClipRect`

CanvasGetPenPosition

```
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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
point	Point	`Point` structure in which to store the current position of the canvas pen.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`CanvasSetPenPosition`, `CanvasDefaultPen`, `CanvasDrawLineTo`

CanvasGetPixel

```
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*** `VAL_DIRECT_TO_SCREEN`***). Sometimes the internal bitmap contains the result of recent drawing operations that have not yet been reflected on the screen.*** `CanvasGetPixel` ***obtains the pixel colors from the internal bitmap, not from the screen.***

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
pixelPoint	Point	`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.

Output

Name	Type	Description
pixelColor	integer	RGB color value of the pixel at the specified point.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakePoint`, `CanvasGetPixels`

CanvasGetPixels

```
int status = CanvasGetPixels (int panelHandle, int controlID, Rect rect,
                              int pixelColors[]);
```

Purpose

Obtains the colors of the pixels in a specific 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*** `VAL_DIRECT_TO_SCREEN`***). Sometimes the internal bitmap contains the result of recent drawing operations that have not yet been reflected on the screen.*** `CanvasGetPixels` ***obtains the pixel colors from the internal bitmap, not from the screen.***

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
rect	Rect	`Rect` structure specifying the location and size of the rectangular area from which to obtain the pixel colors. 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.

Output

Name	Type	Description
pixelColors	integer array	Array of RGB color values of the pixels in the specified rectangle. Refer to discussion below.

Parameter Discussion

The total number of elements in the **pixelColors** array must be equal to rect.height * rect.width. The pixel color values are stored in row-major order. For example, consider a **rect** with the following values.

Values for a Rect Example

Rect Parameter	Value
rect.top	50
rect.left	60
rect.height	20
rect.width	15

The color of pixel {x = 65, y = 58} of this **rect** is stored in a pixel array at the following index:

$$\begin{aligned} &= (y - \text{rect.top}) \times \text{rect.width} + (x - \text{rect.left}) \\ &= (58 - 50) \times 15 + (65 - 60) \\ &= 125 \end{aligned}$$

When using a rect.width of `VAL_TO_EDGE`, substitute the following for rect.width in the preceding formula:

(*total width of canvas*) – rect.left

Return Value

Name	Type	Description
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. Which colors result from the inversion depends on the operating system. If you invert the same rectangle twice, you are guaranteed to get the original colors back.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
rect	Rect	`Rect` structure that specifies the location and size of the rectangular area in which to invert the colors. Use `VAL_ENTIRE_OBJECT` to specify the entire canvas.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakeRect`

CanvasScroll

```
int status = CanvasScroll (int panelHandle, int controlID, Rect scrollRect,
                           int scrollAmountInXDirection,
                           int scrollAmountInYDirection);
```

Purpose

Scrolls the contents of a specific 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 rectangular area are not affected.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
scrollRect	Rect	`Rect` structure that specifies the location and size of the rectangular area to scroll. Use `VAL_ENTIRE_OBJECT` to specify the entire canvas.
scrollAmountIn XDirection	integer	Amount to scroll horizontally.
scrollAmountIn YDirection	integer	Amount to scroll vertically.

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

```
int status = CanvasSetClipRect (int panelHandle, int controlID,
                                Rect clipRect);
```

Purpose

Sets the clipping rectangle for the canvas control. Canvas drawing operations do not extend beyond the clipping rectangle. Any drawing outside the clipping rectangle has no effect. Exception: `CanvasClear` is not limited to the clipping rectangle.

Changing the clipping rectangle does not affect the current contents of the canvas.

In the initial state for a canvas control, clipping is disabled.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
clipRect	Rect	`Rect` structure specifying into the location and size of the clipping rectangle. To disable clipping, set the height and width of **clipRect** to zero, or use `VAL_EMPTY_RECT`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
point	Point	`Point` structure specifying the new position of the canvas pen.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`CanvasGetPenPosition`, `CanvasDefaultPen`, `CanvasDrawLineTo`

CanvasStartBatchDraw

```
int nestingDepth = CanvasStartBatchDraw (int panelHandle, int controlID);
```

Purpose

Allows you to increase the drawing performance of the canvas control. In general, use `CanvasStartBatchDraw` whenever you want to make two or more consecutive calls to canvas drawing functions. Match each call to `CanvasStartBatchDraw` with a call to `CanvasEndBatchDraw`.

Before LabWindows/CVI performs drawing operations, it invokes certain operating system functions to prepare for drawing on the particular canvas. Without batch drawing, LabWindows/CVI must call these system functions for each canvas drawing operation. With batch drawing, LabWindows/CVI calls the system functions only once for all of the drawing operations between `CanvasStartBatchDraw` and the matching `CanvasEndBatchDraw`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
nestingDepth	integer	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 to incorporate a sequence of drawing operations on the same canvas control into one batch:

```
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 can 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 can 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.

Do not access a canvas control from other threads while batch drawing is in effect for the control.

Note ***If the*** `ATTR_DRAW_POLICY` ***attribute for the canvas control is*** `VAL_UPDATE_IMMEDIATELY`***, no update to the screen occurs until you end the batch. Also, changing values of the*** `ATTR_DRAW_POLICY` ***and*** `ATTR_OVERLAP_POLICY` ***attributes during a batch draw has no effect until after you end the batch.***

See Also

`CanvasEndBatchDraw`

CanvasUpdate

```
int status = CanvasUpdate (int panelHandle, int controlID, Rect updateArea);
```

Purpose

Immediately updates on the screen the contents of the canvas control within a specific 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 `VAL_DIRECT_TO_SCREEN`. Maintaining the internal bitmap ensures that the canvas is redrawn correctly when it is exposed.

`CanvasUpdate` copies the content of the rectangular area in the internal bitmap to the canvas control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
updateArea	Rect	`Rect` structure specifying the location and size of the rectangular to update from the internal bitmap. Use `VAL_ENTIRE_OBJECT` to specify the entire canvas control.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`MakeRect`

CheckListItem

```
int status = CheckListItem (int panelHandle, int controlID, int itemIndex,
                            int checked);
```

Purpose

Places a check by, or removes a check from, a specific list item. It applies only to list boxes for which the check mode attribute is enabled.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index of the item in the list.
checked	integer	Specifies whether or not to place a check by the list item.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

ClearAxisItems

```
int status = ClearAxisItems (int panelHandle, int controlID, int axis);
```

Purpose

Deletes all string/value pairs from the list of label strings for a graph or strip chart axis.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

axis	integer	Specifies the axis from which to delete all string/value pair(s). Valid values:
		`VAL_XAXIS`
		`VAL_LEFT_YAXIS`
		`VAL_RIGHT_YAXIS` (graphs only)

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`InsertAxisItem`, `ReplaceAxisItem`, `DeleteAxisItem`, `GetNumAxisItems`

ClearListCtrl

```
int status = ClearListCtrl (int panelHandle, int controlID);
```

Purpose

Clears all list items from a list control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`DeleteListItem`

ClearStripChart

```
int status = ClearStripChart (int panelHandle, int controlID);
```

Purpose

Clears all traces from a strip chart control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

ClipboardGetBitmap

```
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. You can pass the ID of the bitmap to any function that accepts a bitmap, such as `CanvasDrawBitmap`.

You can discard the bitmap by passing its ID to `DiscardBitmap`.

Note ***Under UNIX,*** `ClipboardGetBitmap` ***accesses only the internal LabWindows/CVI clipboard, not the host system clipboard.***

Parameters

Output

Name	Type	Description
bitmapID	integer	ID that serves as a handle to the bitmap copied from the clipboard. `NULL` if no bitmap exists on the clipboard. If you not want a copy of the bitmap, pass `NULL`.
available	integer	`1` if a bitmap is available on the system clipboard, `0` otherwise. You can pass `NULL` for this parameter.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ClipboardPutBitmap`, `ClipboardGetText`, `GetBitmapData`, `SetCtrlBitmap`, `PlotBitmap`, `CanvasDrawBitmap`, `DiscardBitmap`

ClipboardGetText

```
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, free it by calling the `free` function.

☞ **Note** ***Under UNIX,*** `ClipboardGetText` ***accesses only the internal LabWindows/CVI clipboard, not the host system clipboard.***

Parameters

Output

Name	Type	Description
text	string	Pointer to the null-terminated string copied from the clipboard. `NULL` if no text exists on the clipboard. If you not want a copy of the text, pass `NULL`.
available	integer	`1` if a text string is available on the system clipboard, `0` otherwise. You can pass `NULL` for this parameter.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ClipboardPutText`, `ClipboardGetBitmap`

ClipboardPutBitmap

```
int status = ClipboardPutBitmap (int bitmapID);
```

Purpose

Copies a bitmap image onto the system clipboard.

Note ***Under UNIX,*** `ClipboardPutBitmap` ***affects only the internal LabWindows/CVI clipboard, not the host system clipboard.***

Parameter

Input

Name	Type	Description
bitmapID	integer	ID of the bitmap object containing the image. You obtain the ID from `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ClipboardGetBitmap`, `ClipboardPutText`, `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `GetCtrlDisplayBitmap`, `GetPanelDisplayBitmap`

ClipboardPutText

```
int status = ClipboardPutText (char text[]);
```

Purpose

Copies a text string onto the system clipboard.

☞ **Note** ***Under UNIX,*** `ClipboardPutText` ***affects only the internal LabWindows/CVI clipboard, not the host system clipboard.***

Parameter

Input

Name	Type	Description
text	string	Null-terminated string.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ClipboardGetText`, `ClipboardPutBitmap`

ConfigurePrinter

```
int status = ConfigurePrinter (char printFile[], int orientation,
                               int imageWidth, int imageHeight, int ejectAfter);
```

Purpose

Sets the orientation, height, width, and eject after print attributes. Refer to Table 3-69, *Graphics and General Hard Copy Attributes*, in Chapter 3, *Programming with the User Interface Library*, for more information on printing attributes.

☞ **Note** `ConfigurePrinter` ***has been superseded by*** `SetPrintAttribute`**.**

Parameters

Input

Name	Type	Description
printFile	string	LabWindows/CVI ignores **printFile** because you specify the print file in the print function call.

orientation	integer	Orientation of the image on the page. Valid Values:
		`1` = `VAL_PORTRAIT`
		`2` = `VAL_LANDSCAPE`

imageWidth	integer	Width of the image in millimeter/10 or `VAL_USE_PRINTER_DEFAULT` or `VAL_INTEGRAL_SCALE`
imageHeight	integer	Height of the image in millimeter/10 or `VAL_USE_PRINTER_DEFAULT` or `VAL_INTEGRAL_SCALE`
ejectAfter	integer	Determines if the next output is ejected from the printer. While **ejectAfter** is set to zero, outputs print on the same page until **ejectAfter** is set to one.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

ConfirmPopup

```
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

Input

Name	Type	Description
title	string	Title of the dialog box.
message	string	Message to display on the dialog box.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Return Codes

Code	Description
`1`	User selected **Yes**.
`0`	User selected **No**.

CreateMetaFont

```
int status = CreateMetaFont (char newMetaFontName[],
                             char existingFontName[], int pointSize, int bold,
                             int italics, int underlined, int strikeout);
```

Purpose

Creates a new metafont 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

Input

Name	Type	Description
newMetaFontName	string	Name to associate with the new metafont.
existingFontName	string	Name of the existing font on which to base the new metafont. The font can be one of the National Instrument fonts, a user-defined font saved by a previous `CreateMetaFont` function call, or a font supplied by the operating system.
pointSize	integer	Point size of the new metafont. Any positive integer value is valid.
bold	integer	Indicates whether the newly created metafont has bold text. `0` = not bold `1` = bold
italics	integer	Indicates whether the newly created metafont has italicized text. `0` = not italics `1` = italics

Name	Type	Description
underlined	integer	Indicates whether the newly created metafont has underlined text. `0` = not underlined `1` = underlined
strikeout	integer	Indicates whether the newly created metafont has strikeout text. `0` = not strikeout `1` = strikeout

Return Value

Name	Type	Description
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

```
int status = DefaultCtrl (int panelHandle, int controlID);
```

Purpose

Restores a control to its default value.

If the control is visible, it updates to reflect its default value. You assign default values to controls in the User Interface Editor or through `SetCtrlAttribute`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DefaultPanel

```
int status = DefaultPanel (int panelHandle);
```

Purpose

Restores all panel controls to their default values.

If the panel is visible, it updates to reflect the new control values. You assign default values to controls in the User Interface Editor or through `SetCtrlAttribute` using `ATTR_DFLT_VALUE`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DeleteAxisItem

```
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

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.

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	Zero-based index of the first item to delete.
numberOfItems	string	Number of items to delete.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`InsertAxisItem`, `ReplaceAxisItem`, `ClearAxisItems`, `GetNumAxisItems`

DeleteGraphPlot

```
int status = DeleteGraphPlot (int panelHandle, int controlID, int plotHandle,
                              int refresh);
```

Purpose

Deletes one or all plots from a graph control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
plotHandle	integer	Handle for the particular plot to delete or `-1` to delete all plots in the graph control.
refresh	integer	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

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You obtain **plotHandle** from one of the following functions:

```
PlotX
PlotY
PlotXY
PlotWaveform
PlotPoint
```

```
PlotText
PlotLine
PlotRectangle
PlotPolygon
PlotOval
PlotArc
PlotIntensity
PlotBitmap
```

If **refresh** is `VAL_DELAYED_DRAW`, the deleted plot remains on the graph until one of the following actions takes place:

- You rescaled the graph.
- You change the size of the plot area.
- You expose the plot area after hiding or overlapping it.
- You set `ATTR_REFRESH_GRAPH` to `1`.
- You call `RefreshGraph`.
- You add another plot to the graph while `ATTR_REFRESH_GRAPH` is `1`.

When one of the above events occurs, LabWindows/CVI redraws the entire plot area.

If **refresh** is `VAL_IMMEDIATE_DRAW`, the plot area is redrawn immediately.

If **refresh** is `VAL_NO_DRAW`, the deleted plot remains on the graph until one of the following actions takes place:

- You rescale the graph.
- You change the size of the plot area.
- You expose the plot area after hiding or overlapping it, and `ATTR_SMOOTH_UPDATE` is `0`.

DeleteImage

```
int status = DeleteImage (int panelHandle, int controlID);
```

Purpose

Removes an image from a picture control and from memory.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DeleteListItem

```
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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index of the first item to delete.
numberOfItems	integer	Number of items to delete. To delete all items from **itemIndex** to the end, enter `-1` for **numberOfItems.**

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ClearListCtrl, InsertListItem, ReplaceListItem`

DeleteTextBoxLine

```
int status = DeleteTextBoxLine (int panelHandle, int controlID,
                                int lineIndex);
```

Purpose

Removes a line of text from a text box control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
lineIndex	integer	Zero-based index of the text box line to delete.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DirSelectPopup

```
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

Input

Name	Type	Description
defaultDirectory	string	Initial directory. If `""` is entered, then the current working directory is used.
title	string	Title of the dialog box.
allowCancel	integer	If non-zero, the user can cancel out of the dialog box. If zero, the user must make a selection.
allowMakeDirectory	integer	If non-zero, the user is allowed to create a new directory.

Output

Name	Type	Description
pathname	string	Buffer in which the user's selection is returned. The buffer must be at least `MAX_PATHNAME_LEN` bytes long.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Return Codes

Code	Description
`0`	`VAL_NO_DIRECTORY_SELECTED`
`1`	`VAL_DIRECTORY_SELECTED`

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

```
int status = DiscardBitmap (int bitmapID);
```

Purpose

Discards a bitmap object.

Parameter

Input

Name	Type	Description
bitmapID	integer	ID of the bitmap object. You obtain the ID from `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `GetCtrlDisplayBitmap`, `GetPanelDisplayBitmap`, `ClipboardGetBitmap`

DiscardCtrl

```
int status = DiscardCtrl (int panelHandle, int controlID);
```

Purpose

Removes a control from a panel and from memory.

You cannot discard a control in the callback function for the control, except when responding to the `EVENT_COMMIT` event. Discarding the control from its own callback function in response to other events might cause unpredictable behavior.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DiscardMenu

```
int status = DiscardMenu (int menuBarHandle, int menuID);
```

Purpose

Removes a menu and its submenus and items from a menubar.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.
menuID	integer	ID for a particular menu within a menubar. The ID must be a constant name, located in the `.uir` header file, that you assign in the User Interface Editor or a value that you obtain from `NewMenu`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DiscardMenuBar

```
int status = DiscardMenuBar (int menuBarHandle);
```

Purpose

Removes a menubar from every panel on which it resides and from memory.

Parameter

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DiscardMenuItem

```
int status = DiscardMenuItem (int menuBarHandle, int menuItemID);
```

Purpose

Removes a menu item from a menu and from memory.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.
menuItemID	integer	ID for a particular menu item within a menubar. The ID must be a constant name, located in the `.uir` file, that you assign in the User Interface Editor or a value you obtain from `NewMenuItem`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DiscardPanel

```
int status = DiscardPanel (int panelHandle);
```

Purpose

Removes a panel from memory and clears it from the screen if visible.

You must call `DiscardPanel` from the thread in which you create the panel.

Parameter

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DiscardSubMenu

```
int status = DiscardSubMenu (int menuBarHandle, int subMenuID);
```

Purpose

Removes a submenu from a menubar and from memory.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`
subMenuID	integer	ID for a particular menu item within a menubar. The ID must be a constant name, located in the `.uir` file, that you assign in the User Interface Editor or a value you obtain from `NewSubMenu`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DisplayImageFile

```
int status = DisplayImageFile (int panelHandle, int controlID,
                               char filename[]);
```

Purpose

Displays the contents of an image file on a picture control.

You can use the following image types:

`.pcx`	Windows and UNIX
`.bmp`, `.dib`, `.rle`, `.ico`	Windows only
`.wmf`	Windows 95/NT only
`.xwd`	UNIX only

To display an image, first create a picture control in the User Interface Editor or with `NewCtrl`. Then call `DisplayImageFile` using the picture control ID.

To delete the image, call `DeleteImage`, 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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
filename	string	Name of the image file that contains the image. If the name is a simple filename with no directory path, 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.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DisplayPCXFile

```
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. The coordinates can be any value between 0 and 32,767. The origin is in the upper left corner directly beneath the title bar before the panel is scrolled. If the DOS Compatibility window has not yet been created, `DisplayPCXFile` creates it.

You can use the following image types:

`.pcx`	Windows and UNIX
`.bmp`, `.dib`, `.rle`, `.ico`	Windows only
`.wmf`	Windows 95/NT only
`.xwd`	UNIX only

Note `DisplayPCXFile` ***has been superseded by*** `DisplayImageFile`***. Repeated use of*** `DisplayPCXFile` ***clutters the panel with redundant controls.***

Parameters

Input

Name	Type	Description
filename	string	Name of the image file that contains the image. If the name is a simple filename with no directory path, 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.
controlTop	integer	Coordinate of the top edge of the image.
controlLeft	integer	Coordinate of the left edge of the image.

Return Value

Name	Type	Description
controlID	integer	Returns the ID you use to specify this control in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

DisplayPanel

```
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 generate events from it. Calling `DisplayPanel` when a panel is already displayed causes the panel to be completely redrawn.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DOSColorToRGB

```
int status = DOSColorToRGB (int lwDOSColor);
```

Purpose

Translates the 16 standard color values from LabWindows/DOS to RGB values as shown in Table 4-5.

Table 4-5. Standard Color Values

LW/DOS	LW/CVI
`0` (black)	`VAL_BLACK= 0x000000L`
`1` (dark blue)	`VAL_DK_BLUE= 0x000080L`
`2` (dark green)	`VAL_DK_GREEN= 0x008000L`
`3` (dark cyan)	`VAL_DK_CYAN= 0x008080L`
`4` (dark red)	`VAL_DK_RED= 0x800000L`
`5` (dark magenta)	`VAL_DK_MAGENTA= 0x800080L`
`6` (brown)	`VAL_DK_YELLOW= 0x808000L`
`7` (gray)	`VAL_LT_GRAY= 0xCCCCCCL`
`8` (dark gray)	`VAL_DK_GRAY= 0x808080L`
`9` (blue)	`VAL_BLUE= 0x0000FFL`
`10` (green)	`VAL_GREEN= 0x00FF00L`
`11` (cyan)	`VAL_CYAN= 0x00FFFFL`
`12` (red)	`VAL_RED= 0xFF0000L`
`13` (magenta)	`VAL_MAGENTA= 0xFF00FFL`
`14` (yellow)	`VAL_YELLOW= 0xFFFF00L`
`15` (white)	`VAL_WHITE= 0xFFFFFFL`

Parameter

Input

Name	Type	Description
lwDOSColor	integer	One of the 16 standard colors from LabWindows/DOS.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

DOSCompatWindow

```
int panelHandle = DOSCompatWindow (void);
```

Purpose

Displays a window that serves as a parent panel for panels you created in a LabWindows for DOS program.

You can use the return value as the **parentPanelHandle** parameter to the `LoadPanel` function. You can use `DOSCompatWindow` with LabWindows for DOS `.uir` files to retain the appearance of the DOS User Interface screen.

`DOSCompatWindow` exists as an aid for converting LabWindows for DOS applications. Do not use it when developing new applications in LabWindows/CVI.

Return Value

Name	Type	Description
panelHandle	integer	Returns the panel handle for the DOS Compatibility Window. A negative number indicates that an error occurred. Refer to Appendix A for error codes.

DuplicateCtrl

```
int newID = DuplicateCtrl (int sourcePanelHandle, int controlID,
                           int destinationPanelHandle,
                           char duplicateLabel[], int controlTop,
                           int controlLeft);
```

Purpose

`DuplicateCtrl` copies an existing control from the source panel to the destination panel and returns a control ID. You use the control ID to reference the control in subsequent function calls.

Parameters

Input

Name	Type	Description
sourcePanelHandle	integer	Handle of the source panel containing the control to duplicate. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
destinationPanelHandle	integer	Specifies the parent panel into which to copy the duplicate control. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
duplicateLabel	string	Label for the duplicate control. Pass `""` for no label. Pass `0` to use the label of the source control.
controlTop	integer	Vertical coordinate at which to place the upper left corner of the control, not including labels.
controlLeft	integer	Horizontal coordinate at which to place the upper left corner of the control, not including labels.

Return Value

Name	Type	Description
newID	integer	Returns the ID you use 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

Duplicates a panel into the specified destination parent panel and returns the duplicate (new) panel handle.

Parameters

Input

Name	Type	Description
destParentPanelHandle	integer	Handle for the parent panel into which to copy the duplicate panel. To make the panel a top-level panel, pass `0`.
originalPanelHandle	integer	Handle of the original panel to duplicate. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
duplicatePanelTitle	string	Title of the duplicate (new) panel. Pass `""` for no title. Pass `0` to use the title of the original panel.
panelTop	integer	Vertical coordinate at which to place the upper left corner of the panel, directly below the title bar. Pass `VAL_KEEP_SAME_POSITION` to keep the same top coordinate as the original panel.
panelLeft	integer	Horizontal coordinate at which to place the upper left corner of the panel, directly below the title bar. Pass `VAL_KEEP_SAME_POSITION` to keep the same left coordinate as the original panel.

Return Value

Name	Type	Description
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 –32,768 to 32,767, or `VAL_AUTO_CENTER` to center the panel.

EmptyMenu

```
int status = EmptyMenu (int menuBarHandle, int menuID);
```

Purpose

Removes all submenus and menu items from a specific menu.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.
menuID	integer	ID for a particular menu within a menubar. The Menu ID should be a constant name, located in the `.uir` header file, that the User Interface Editor generates or a value you obtain from `NewMenu`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

EmptyMenuBar

```
int status = EmptyMenuBar (int menuBarHandle);
```

Purpose

Removes all menus and menu items from a specific menubar but retains the menubar handle in memory.

Parameter

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

FakeKeystroke

```
int status = FakeKeystroke (int keyCode);
```

Purpose

Simulates a keystroke by posting a keystroke event to the currently active panel.

☞ **Note** ***The order in which you receive the keystroke event in relation to other events is not defined, so you must use*** `FakeKeystroke` ***with care.***

Parameters

Input

Name	Type	Description
keyCode	integer	You form key codes by a logical OR operation on values representing a modifier key and either an ASCII character or a virtual key. Refer to Table 3-7 in Chapter 3 for modifier and virtual keys.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

FileSelectPopup

```
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

Input

Name	Type	Description
defaultDirectory	string	Initial directory. If `""` is entered, the current working directory is used.
defaultFileSpec	string	String that specifies which files to display. For example, `"*.c"` causes all files with the extension `.c` to be displayed.
fileTypeList	string	List of file types, separated by semicolons, to display in the File Type List of the File Select Pop-up when **restrictExtension** is FALSE. For example, `"*.c; *.h"` allows the user to select `"*.c"` or `"*.h"` from the File Type List. The All Files, `*.*`, option is always available.
title	string	Title of the dialog box.

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.

Name	Type	Description
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.

Output

Name	Type	Description
pathName	string	Buffer in which the user's selection is returned. The buffer must be at least `MAX_PATHNAME_LEN` bytes long.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Return Codes

Code	Description
`0`	`VAL_NO_FILE_SELECTED`
`1`	`VAL_EXISTING_FILE_SELECTED`
`2`	`VAL_NEW_FILE_SELECTED`

Parameter Discussion

The **defaultFileSpec** appears in the file name control when you initially display the pop-up. 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 must declare the **pathName** string to be a least `MAX_PATHNAME_LEN` bytes long.

FontSelectPopup

```
int status = FontSelectPopup (char title[], char sampleText[],
                              int monospacedFontsOnly,
                              char typefaceName[], int *bold,
                              int *underline, int *strikeOut,
                              int *italic, int *justification,
                              int *textColor, int *fontSize,
                              int minimumFontSize, int maximumFontSize,
                              int showDefaultButton, int allowMetaFonts);
```

Purpose

Opens a dialog box that allows the user to specify font settings.

Parameters

Input

Name	Type	Description
title	string	Title to be displayed on the dialog box. The maximum length is 255 characters.
sampleText	string	Sample text to be displayed in the font select pop-up as demonstration of how the settings affect the appearance of text.
monospacedFontsOnly	integer	If nonzero, the user can select only monospaced, fixed width, fonts. If zero, the user can select any font.
minimumFontSize	integer	Minimum value allowed in the Font Size control.
maximumFontSize	integer	Maximum value allowed in the Font Size control.
showDefaultButton	integer	If zero, the **Default** button is hidden. If non-zero, the **Default** button appears. When the user presses the **Default** button, the controls on the pop-up are set to the values you specified in your most recent call to `SetFontPopupDefaults`.
allowMetaFonts	integer	If zero, the National Instruments supplied metafonts are not listed in the typeface selection ring. If nonzero, the metafonts are listed.

Input/Output

Name	Type	Description
typefaceName	string	On input, this buffer contains the typeface name, for example, "Courier," that initially appears in the selection ring. On output, this buffer contains the typeface name the user selects. 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.
bold	integer	On input, the value that initially appears in the Bold checkbox. On output, the final value in the Bold checkbox. Pass `0` to hide the Bold checkbox.
underline	integer	On input, the value that initially appears in the Underline checkbox. On output, the final value in the Underline checkbox. Pass `0` to hide the Underline checkbox.
strikeOut	integer	On input, the value that initially appears in the StrikeOut checkbox. On output, the final value in the StrikeOut checkbox. Pass `0` to hide the StrikeOut checkbox.
italic	integer	On input, the value that initially appears in the Italic checkbox. On output, the final value in the Italic checkbox. Pass `0` to hide the Italic checkbox.
justification	integer	On input, the value that initially appears in the Justification ring control. On output, the final value in the Justification ring control. Pass `0` to hide the Justification ring control.

Name	Type	Description
textColor	integer	On input, the value that initially appears in the Text Color control. On output, the final value in the Text Color control. Pass `0` to hide the Text Color control.
fontSize	integer	On input, the value that initially appears in the Font Size control. On output, the final value in the Font Size control. Pass `0` to hide the Font Size control.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Return Codes

Code	Description
`0`	User canceled out of dialog box.
`1`	User modified the settings.

Parameter Discussion

The valid values for justification are:

```
VAL_LEFT_JUSTIFIED
VAL_RIGHT_JUSTIFIED
VAL_CENTER_JUSTIFIED
```

textColor is an RGB value. An RGB value is an integer in the hexadecimal format `0x00RRGGBB`, where `RR`, `GG`, and `BB` are the respective red, green, and blue components of the color value.

Specify **fontSize** in units of points.

If the user cancels out of the dialog box or an error occurs, the functions modifies none of the Input/Output parameters.

See Also

`SetFontPopupDefaults`

GenericMessagePopup

```
int button = GenericMessagePopup (char title[], char message[],
                           char buttonLabel1[], char buttonLabel2[],
                           char buttonLabel3[], char responseBuffer[],
                           int maxResponseLength, int buttonAlignment,
                           int activeControl, int enterButton,
                           int escapeButton);
```

Purpose

Displays a dialog box with a defined message and optionally accepts a response string.

You can use up to three buttons and specify their labels. `GenericMessagePopup` returns a value indicating which button the user pressed.

Parameters

Input

Name	Type	Description
title	string	Title of the dialog box.
message	string	Message displayed on the dialog box.
buttonLabel1	string	Label on button 1.
buttonLabel2	string	Label on button 2. To hide buttons 2 and 3, pass `0` to Button Label 2.
buttonLabel3	string	Label on button 3. To hide button 3, pass `0` to Button Label 3.
maxResponseLength	integer	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 NUL byte.
buttonAlignment	integer	Selects the location of the buttons. A non-zero value aligns the buttons along the right-hand side of the dialog box. A value of zero aligns the buttons along the bottom of the dialog box.

Name	Type	Description

activeControl	integer	Selects one of the buttons or the input string as the active control. The active control is the control that accepts keystrokes. The following values apply to **activeControl**:
		`VAL_GENERIC_POPUP_BTN1`
		`VAL_GENERIC_POPUP_BTN2`
		`VAL_GENERIC_POPUP_BTN3`
		`VAL_GENERIC_POPUP_INPUT_STRING`

enterButton	integer	Selects which button has <Enter> as its shortcut key. If no button is to have <Enter> as its shortcut key, pass `VAL_GENERIC_POPUP_NO_CTRL`.
escapeButton	integer	Selects which button has <Esc> as its shortcut key. If no button is to have <Esc> as its shortcut key, pass `VAL_GENERIC_POPUP_NO_CTRL`.

Output

Name	Type	Description
responseBuffer	string	Buffer in which to store the user's response. The buffer must be large enough to hold **maxResponseLength** bytes plus one ASCII NUL byte. To hide the input string, pass `0`.

Return Value

Name	Type	Description
button	integer	Returns an indication of which button the user clicked on. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

Return Codes

Code	Description
`1`	`VAL_GENERIC_POPUP_BTN1`
2	`VAL_GENERIC_POPUP_BTN2`
3	`VAL_GENERIC_POPUP_BTN3`

Get3dBorderColors

```
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 three-dimensional. The colors `Get3dBorderColors` returns are similar to the colors Windows 95/NT uses for drawing 3-dimensional objects.

Parameters

Input

Name	Type	Description
baseColor	integer	RGB value for the color of an object.

Output

Name	Type	Description
highlightColor	integer	RGB value for the color that indicates the edges of the object that are in the most direct light.
lightColor	integer	RGB value for the color that indicates the transition between the highlight color and the base color of the object.
shadowColor	integer	RGB value for the color that indicates the edges of the object that are angled away from the light.
darkShadowColor	integer	RGB value for the color that indicates the edges of the object that are angled farthest away from the light.

Parameter Discussion

You can pass `NULL` for any of the output parameters.

Currently, the **lightColor** is always the same as the **baseColor**, as is the case under Windows 95/NT.

Currently, the **darkShadowColor** is always black, as is the case under Windows 95/NT.

GetActiveCtrl

```
int activeCtrl = GetActiveCtrl (int panelHandle);
```

Purpose

Obtains the ID of the active control on a specific panel.

The active control is the control that receives keyboard events when the panel is the active panel.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Return Value

Name	Type	Description
activeCtrl	integer	Returns the control ID of the active control. Refer to Appendix A for error codes.

GetActiveGraphCursor

```
int status = GetActiveGraphCursor (int panelHandle, int controlID,
                                   int *activeCursorNumber);
```

Purpose

Obtains the active cursor on a graph control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
activeCursorNumber	integer	Returns the number of the active cursor. The value ranges from 1 to the number of cursors for the graph. You set the number of cursors in the User Interface Editor or through `SetCtrlAttribute`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetActivePanel

```
int activePanel = GetActivePanel (void);
```

Purpose

Obtains the handle of the active panel. The active panel is the panel that receives keyboard events.

Return Value

Name	Type	Description
activePanel	integer	Returns the handle of the active panel. Refer to Appendix A for error codes.

GetAxisItem

```
int status = GetAxisItem (int panelHandle, int controlID, int axis,
                          int itemIndex, char itemLabel[],
                          double *itemValue);
```

Purpose

Retrieves the string/value pair at a specific index in the list of label strings for a graph or strip chart axis.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

axis	integer	Specifies the axis from which to retrieve the selected string/value pair. Valid values:
		`VAL_XAXIS`
		`VAL_LEFT_YAXIS`
		`VAL_RIGHT_YAXIS` (graphs only)

itemIndex	integer	Zero-based index into the list of label strings.

Output

Name	Type	Description
itemLabel	string	Buffer in which the label string is returned.
itemValue	double-precision	Value associated with the label string.

Return Value

Name	Type	Description
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 can pass `NULL` for either of the output parameters.

See Also

`InsertAxisItem`, `GetNumAxisItems`, `GetAxisItemLabelLength`

GetAxisItemLabelLength

```
int status = GetAxisItemLabelLength (int panelHandle, int controlID,
                           int axis, int itemIndex, int *length);
```

Purpose

Obtains the number of characters in a label string for a graph or strip chart axis. You specify the label string by its index in the list of string/value pairs for that axis.

The length returned does not include the terminating NUL byte.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

axis	integer	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)

itemIndex	integer	Zero-based index into the list of label strings.

Output

Name	Type	Description
length	string	Length of the selected label string. Excludes the terminating NUL byte.

Return Value

Name	Type	Description
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 can pass `NULL` for either of the output parameters.

See Also

`InsertAxisItem`, `GetNumAxisItems`, `GetAxisItem`

GetAxisRange

```
int status = GetAxisRange (int panelHandle, int controlID, int *xAxisScaling,
                           double *xmin, double *xmax, int *yAxisScaling,
                           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.

`GetAxisRange` does not work on the right y-axis and is therefore obsolete. It is recommended that you use `GetAxisScalingMode` instead. To obtain the x-offset and x-increment for a strip chart, use the `ATTR_XAXIS_GAIN` and `ATTR_XAXIS_OFFSET` attributes.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
xAxisScaling	integer	Current scaling mode of the x-axis. `0L` = `VAL_MANUAL` `1L` = `VAL_AUTOSCALE` (graphs only)
xmin	double-precision	Current minimum range of the x-axis.
xmax	double-precision	Current maximum range of the x-axis.
yAxisScaling	integer	Current scaling mode of the y-axis. `0L` = `VAL_MANUAL` `1L` = `VAL_AUTOSCALE` (graphs only)
ymin	double-precision	Current minimum range of the y-axis.
ymax	double-precision	Current maximum range of the y-axis.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

xAxisScaling

`VAL_MANUAL`	The x-axis is manually scaled, and **xmin** and **xmax** define its range.
`VAL_AUTOSCALE`	The x-axis is autoscaled. **xmin** and **xmax** are not returned. Autoscaling is not allowed for strip charts.

yAxisScaling

`VAL_MANUAL`	The y-axis is manually scaled, and **ymin** and **ymax** define its range.
`VAL_AUTOSCALE`	The y-axis is autoscaled. **ymin** and **ymax** are not returned. Autoscaling is not allowed for strip charts.

GetAxisScalingMode

```
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.

`GetAxisScalingMode` is not valid for the x-axis of a strip chart. To obtain the x-offset and x-increment for a strip chart, use `GetCtrlAttribute` with the `ATTR_XAXIS_OFFSET` and `ATTR_XAXIS_GAIN` attributes.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from the `NewCtrl` or `DuplicateCtrl`.
axis	integer	Specifies for which axis to obtain the mode and range. Valid values: `VAL_XAXIS` (graphs only) `VAL_LEFT_YAXIS` (graphs and strip charts) `VAL_RIGHT_YAXIS` (graphs only)

Output

Name	Type	Description
axisScaling	integer	Scaling mode used for the axis. Refer to Table 4-6.
min	double-precision	Current minimum value on the axis.
max	double-precision	Current maximum value on the axis.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

Table 4-6 lists the values of **axisScaling**.

Table 4-6. AxisScaling Values

Valid Value	Description
`VAL_MANUAL`	Axis is manually scaled, and **min** and **max** define its range.
`VAL_AUTOSCALE`	Axis is autoscaled. **min** and **max** are not used. `VAL_AUTOSCALE` is not valid for strip charts.
`VAL_LOCK`	Axis is manually scaled using the current, possibly auto scaled, minimum and maximum values on the axis. `VAL_LOCK` is not valid for strip charts.

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 can pass `NULL` for any of the output parameters.

See Also

`SetAxisScalingMode`

GetBitmapData

```
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 to pass, and then allocate the buffers.
- Call `AllocBitmapData`.

Parameters

Input

Name	Type	Description
bitmapID	integer	ID of the bitmap object containing the image. You obtain the ID from `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`.

Output

Name	Type	Description
bytesPerRow	integer	Number of bytes on each scan line of the image.
pixelDepth	integer	Number of bits per pixel.
width	integer	Width of the image, in pixels.
height	integer	Height of the image, in pixels.
colorTable	integer array	Array of RGB color values, or `NULL` if **pixelDepth** is greater than eight.
bits	unsigned char array	Array of bits that determine the colors to display on each pixel in the image.
mask	unsigned char array	Array containing one bit per pixel in the image. Each bit specifies whether to actually draw the pixel. Can be `NULL`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

If no image exists, 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 eight or less, the **bits** array is filled with indices into the **colorTable** array, and the number of entries in the **colorTable** array is two raised to the power of the **pixelDepth** parameter. If **pixelDepth** is greater than eight, 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 is always in the least significant byte. On little-endian platforms, for example, Intel processors, `BB` is at the lowest memory address. On big-endian platforms, for example, Motorola processors, `BB` is at the highest address. Notice that this byte ordering scheme differs from the byte ordering scheme 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.

In the **mask** array, a bit value of `1` indicates that the pixel is drawn. `0` indicates that the pixel is not drawn. An exception applies when the **pixelDepth** is `1`. In this case, the pixels that have a **bits** value of `1`, called foreground pixels, are always drawn, and the **mask** affects only the pixels that have a value of `0`, called background pixels.

You must pad each row of the **mask** to the nearest even-byte boundary. For example, if the width of the image is 21 pixels, then each row of the mask must have 32 bits, in other words, four bytes, of data.

A mask is useful for achieving transparency.

You can pass `NULL` for any of the output parameters.

See Also

`NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `GetCtrlDisplayBitmap`, `GetPanelDisplayBitmap`, `ClipboardGetBitmap`, `GetBitmapInfo`, `AllocBitmapData`, `SetBitmapData`

GetBitmapFromFile

```
int status = GetBitmapFromFile (char fileName[], int *bitmapID);
```

Purpose

Reads a bitmap image from a file and creates a bitmap object. You can pass the bitmap ID to any function that accepts a bitmap, such as `CanvasDrawBitmap` or `ClipboardPutBitmap`.

You can discard the bitmap object by passing the ID to `DiscardBitmap`.

You can use the following image types:

`.pcx`	Windows and SPARCstation
`.bmp`, `.dib`, `.rle`, `.ico`	Windows only
`.wmf`	Windows 95/NT only
`.xwd`	SPARCstation only

Parameters

Input

Name	Type	Description
fileName	string	Pathname of the file that 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.

Output

Name	Type	Description
bitmapID	integer	ID that serves as a handle to the bitmap object.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ClipboardPutBitmap`, `GetBitmapData`, `SetCtrlBitmap`, `PlotBitmap`, `CanvasDrawBitmap`, `DiscardBitmap`

GetBitmapInfo

```
int status = GetBitmapInfo (int bitmapID, int *colorSize,
                           int *bitsSize, int *maskSize);
```

Purpose

Obtains size information about the image associated with a bitmap. You can use this information to allocate the buffers you pass to `GetBitmapData`.

As an alternative to this function, you can call `AllocBitmapData`, which allocates the buffers for you.

Parameters

Input

Name	Type	Description
bitmapID	integer	ID of the bitmap object containing the image. You obtain the ID from `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`.

Output

Name	Type	Description
colorSize	integer	Number of bytes in the image color table; `0` if the pixel depth of the image is greater than eight.
bitsSize	integer	Number of bytes in the image bitmap.
maskSize	integer	Number of bytes in the image mask; `0` if no mask exists.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You can pass `NULL` for any of the output parameters.

See Also

`GetBitmapData`, `AllocBitmapData`

GetCtrlAttribute

```
int status = GetCtrlAttribute (int panelHandle, int controlID,
                               int controlAttribute, void *attributeValue);
```

Purpose

Obtains the value of a control attribute.

Control attributes have differing data types and differing valid ranges. A list of attributes, their data types and their valid values appears in Tables 3-9 to 3-44 in Chapter 3, *Programming with the User Interface Library*.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
controlAttribute	integer	Selects a control attribute. Many control attributes are specific to one kind of control, while others are used with all or many different controls. Refer to Tables 3-9 to 3-44 in Chapter 3 for a complete listing of control attributes.

Output

Name	Type	Description
attributeValue	void *	Current value of control attribute. Refer to Tables 3-9 to 3-44 in Chapter 3 for a complete listing of control attributes.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetCtrlBitmap

```
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. You can pass the bitmap ID 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 you create 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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
imageID	integer	For picture rings, the zero-based index of an image in the ring. For graphs, this argument is the **plotHandle** you obtain from `PlotBitmap`. For picture controls, picture buttons, and canvas controls, this argument is ignored.

Output

Name	Type	Description
bitmapID	integer	ID that serves as a handle to the bitmap object.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ClipboardPutBitmap`, `GetBitmapData`, `SetCtrlBitmap`, `PlotBitmap`, `CanvasDrawBitmap`, `DiscardBitmap`

GetCtrlBoundingRect

```
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 of the control's bounding rectangle.

The bounding rectangle encloses the body of the control and all of labels of the control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
top	integer	Returns the top coordinate of the control's bounding rectangle.
left	integer	Returns the left coordinate of the control's bounding rectangle.
height	integer	Returns the vertical size of the control's bounding rectangle.
width	integer	Returns the horizontal size of the control's bounding rectangle.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You can pass `NULL` for any output parameter.

The range of the **top** and **left** parameters is –32,768 to 32,767. The range of the **height** and **width** parameters is 1 to 32,767. The origin (0,0) is at the upper left corner of the panel, before the panel is scrolled, directly below the title bar.

GetCtrlDisplayBitmap

```
int status = GetCtrlDisplayBitmap (int panelHandle, int controlID,
                                   int includeLabel, int *bitmapID);
```

Purpose

Creates a bitmap object containing a screen shot image of the current appearance of a control. You can pass the bitmap ID 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 `DiscardBitmap`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
includeLabel	integer	If nonzero, the control label is included in the image.

Output

Name	Type	Description
bitmapID	integer	ID that serves as a handle to the bitmap object.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ClipboardPutBitmap`, `GetBitmapData`, `GetPanelDisplayBitmap`, `SetCtrlBitmap`, `PlotBitmap`, `CanvasDrawBitmap`, `DiscardBitmap`, `GetScaledCtrlDisplayBitmap`

GetCtrlIndex

```
int status = GetCtrlIndex (int panelHandle, int controlID, int *itemIndex);
```

Purpose

Returns the zero-based index of the currently selected item in a list box or ring control.

To obtain the value of the currently selected item, call `GetCtrlVal`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
itemIndex	integer	Returns the zero-based current index of the currently selected item. Returns `-1` if the list control has no items.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetCtrlVal

```
int status = GetCtrlVal (int panelHandle, int controlID, void *value);
```

Purpose

Obtains the current value of a control.

When the control ID is for a list box or ring control, `GetCtrlVal` returns the value of the currently selected list item. To obtain the index of the selected list item, use `GetCtrlIndex`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
value	void *	Returns the control value. The data type of **value** must match the data type of the control.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetCursorAttribute

```
int status = GetCursorAttribute (int panelHandle, int controlID,
                                 int cursorNumber, int cursorAttribute,
                                 int *attributeValue);
```

Purpose

Obtains the value of a graph cursor.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
cursorNumber	integer	Identifies the cursor. Can be `1` to the number of cursors for the graph. You set the number of cursors in the User Interface Editor or through `SetCtrlAttribute`.

cursorAttribute	integer	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`

Output

Name	Type	Description
attributeValue	integer	Current value of the cursor attribute. Refer to Tables 3-55 to 3-60 in Chapter 3 for a complete listing of cursor attributes.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetCVITaskHandle

```
int taskHandle = GetCVITaskHandle (void);
```

☞ **Note** ***Only the Windows 3.1 version of LabWindows/CVI supports*** `GetCVITaskHandle`**.**

Purpose

Returns the Windows 3.1 task handle associated with the LabWindows/CVI application. You can use this number in a DLL that requires the LabWindows/CVI Windows task handle.

For Windows 95/NT, use the Windows API function `GetCurrentProcessID`.

Return Value

Name	Type	Description
taskHandle	integer	Windows 3.1 task handle associated with the LabWindows/CVI application.

GetCVIWindowHandle

```
int windowHandle = GetCVIWindowHandle (void);
```

☞ **Note** ***Only the Windows versions of LabWindows/CVI support*** `GetCVIWindowHandle`**.**

Purpose

Returns the Windows window handle associated with the LabWindows/CVI application. You can use this number as the `hwnd` parameter for the Windows `PostMessage` function to post a message to your LabWindows/CVI program from a DLL or another application.

In a multithreaded application, `GetCVIWindowHandle` returns the window handle associated with the current thread.

Return Value

Name	Type	Description
windowHandle	integer	Windows handle associated with the LabWindows/CVI application.

GetGlobalMouseState

```
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** return the position of the mouse relative to the top, left corner of the screen.

Parameters

Output

Name	Type	Description
panelHandle	integer	Handle of the panel that the mouse is over. `0` if the mouse is not over a panel.
xCoordinate	integer	X-coordinate of the mouse cursor relative to the left edge of the screen.
yCoordinate	integer	Y-coordinate of the mouse cursor relative to the top of the screen.
leftButtonDown	integer	`0` if the left button is up. `1` if the left button is down.
rightButtonDown	integer	`0` if the right button is up. `1` if the right button is down.
keyModifiers	integer	State of the keyboard modifiers, in other words, the <Ctrl> and <Shift> 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`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You can pass `NULL` (`0`) in place of any of the output parameters.

See Also

`GetRelativeMouseState`

GetGraphCursor

```
int status = GetGraphCursor (int panelHandle, int controlID,
                             int cursorNumber, double *x, double *y);
```

Purpose

Obtains the current position of a specific graph cursor.

The position is relative to the current range of the x- and y-axes.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
cursorNumber	integer	Identifies the cursor. Can be `1` to the number of cursors for the graph. You set the number of cursors in the User Interface Editor or through `SetCtrlAttribute`.

Output

Name	Type	Description
x	double-precision	Returns the x-coordinate of the cursor position.
y	double-precision	Returns the y-coordinate of the cursor position.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetGraphCursorIndex

```
int status = GetGraphCursorIndex (int panelHandle, int controlID,
                                  int cursorNumber, int *plotHandle,
                                  int *arrayIndex);
```

Purpose

Obtains the plot handle and array index of the point to which the cursor is currently attached.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
cursorNumber	integer	Identifies the cursor. Can be `1` to the number of cursors for the graph. You set the number of cursors in the User Interface Editor or through `SetCtrlAttribute`.

Output

Name	Type	Description
plotHandle	integer	Returns the ID of the plot to which the cursor is attached. If the cursor is not attached to a plot, returns `-1`.
arrayIndex	integer	Returns 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, returns `-1`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetImageBits

```
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, and then allocate the buffers, or
- Call `AllocImageBits`.

The following control types can contain images: picture controls, picture rings, picture buttons, and graph controls.

You can use this function on images you create using `DisplayImageFile`, `InsertListItem`, `ReplaceListItem`, `PlotBitmap`, or `SetImageBits`, or `SetCtrlAttribute` with the `ATTR_IMAGE_FILE` attribute.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
imageID	integer	For a picture ring, the zero-based index of an image in the ring. For a graph, the **plotHandle** you obtain from `PlotBitmap`. For picture controls and buttons, this is ignored.

Output

Name	Type	Description
rowBytes	integer	Number of bytes on each scan line of the image.
depth	integer	Number of bits per pixel.
width	integer	Width of the image, in pixels.
height	integer	Height of the image, in pixels.
colorTable	integer array	Array of RGB color values.
bitmap	unsigned char array	Array of bits that determine the colors to be displayed on each pixel in the image.
mask	unsigned char array	Array containing one bit per pixel in the image. Each bit specifies whether to actually draw the pixel.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

If no image exists, the **width** and **height** parameters are set to `-1`. If the image was originally rendered from a Windows metafile (`.wmf`), this function determines the size of the bitmap as follows:

- If the image is 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 two 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 eight or less, the **bits** array is filled with indices into the **colorTable** array, and the number of entries in the **colorTable** array is two raised to the power of the **pixelDepth** parameter. If **pixelDepth** is greater than eight, 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 is in the least significant byte. On little-endian platforms, for example, Intel processors, `BB` is at the lowest memory address. On big-endian platforms, for example, Motorola processors, `BB` is at the highest address. Notice that this byte ordering scheme differs from the byte ordering scheme 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.

In the **mask** array, a bit value of `1` indicates that the pixel is drawn. `0` indicates that the pixel is not drawn. An exception applies when the **pixelDepth** is `1`. In this case, the pixels that have a **bits** value of `1`, called foreground pixels, are always drawn, and the **mask** affects only the pixels that have a value of `0`, called background pixels.

You must pad each row of the **mask** to the nearest even-byte boundary. For example, if the width of the image is 21 pixels, then each row of the mask must have 32 bits, in other words, four bytes, of data.

A mask is useful for achieving transparency.

You can pass `NULL` for any of the output parameters.

See Also

`GetImageInfo`, `AllocImageBits`, `SetImageBits`

GetImageInfo

```
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. You use the information to allocate the buffers you pass to `GetImageBits`.

The following control types can contain images: picture controls, picture rings, picture buttons, and graph controls.

As an alternative to this function, you can call `AllocImageBits`, which allocates the buffers for you.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
imageID	integer	For a picture ring, the zero-based index of an image in the ring. For a graph, the **plotHandle** you obtain from `PlotBitmap`. For picture controls and buttons, this is ignored.

Output

Name	Type	Description
colorSize	integer	Number of bytes in the image color table. `-1` if no image exists.
bitmapSize	integer	Number of bytes in the image bitmap. `-1` if no image exists.
maskSize	integer	Number bytes in the image mask. `0` if the image has no mask. `-1` if no image exists.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You can pass `NULL` for any of the output parameters.

See Also

`GetImageBits`

GetIndexFromValue

```
int status = GetIndexFromValue (int panelHandle, int controlID,
                                int *index, ...);
```

Purpose

Returns the zero-based index of the first list item that has the value you specify.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
itemValue	Depends on the type of the control.	Specifies the value to search for among the list items.

Output

Name	Type	Description
index	integer	Returns the zero-based index of the first list item that has the value you specify. Returns `-1` if none of the list items have the specified value.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetLabelFromIndex

```
int status = GetLabelFromIndex (int panelHandle, int controlID,
                                int itemIndex, char itemLabel[]);
```

Purpose

Returns the label of a specific list item.

For picture ring controls, this function returns an empty string.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list.

Output

Name	Type	Description
itemLabel	string	Returns the label of the list item you specify.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetLabelLengthFromIndex

```
int status = GetLabelLengthFromIndex (int panelHandle, int controlID,
                                      int itemIndex, int *length);
```

Purpose

Returns the label length of a specific list item.

For picture ring controls, this function returns zero.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list.

Output

Name	Type	Description
length	integer	Returns the number of characters in the label of the list item you specify.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetListItemImage

```
int status = GetListItemImage (int panelHandle, int controlID, int itemIndex,
                               int *image);
```

Purpose

Returns a value corresponding to the image state of a specific 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, refer to `GetImageBits`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list.

Output

Name	Type	Description

image	integer	Returns the value corresponding to the image state of the list item you specify. The possible image states are:
		`0` = `VAL_NO_IMAGE`
		`1` = `VAL_FOLDER`
		`2` = `VAL_OPEN_FOLDER`
		`3` = `VAL_CURRENT_FOLDER`

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetMenuBarAttribute

```
int status = GetMenuBarAttribute (int menuBarHandle, int menuOrMenuItemID,
                                  int menuBarAttribute, void *attributeValue);
```

Purpose

This function returns the value of the specified menubar attribute.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	ID that you assign to the menu or menu item in the User Interface Editor, located in the `.uir` header file, or that you obtain from `NewMenu`, `NewSubMenu`, or `NewMenuItem.`
menuOrMenuItemID	integer	ID that you assign to the menu or menu item in the User Interface Editor or that you obtain from `NewMenu`, `NewSubMenu`, or `NewMenuItem`. If the attribute corresponds to the entire menubar, pass `0` as this parameter.
menuBarAttribute	integer	A particular menubar attribute. Refer to Table 3-6 in Chapter 3 for a complete listing of menubar attributes.

Output

Name	Type	Description
attributeValue	void *	Current value of the menubar attribute. Refer to Table 3-6 in Chapter 3 for a complete listing of all menubar attributes, their data types, and their valid values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetMouseCursor

```
int status = GetMouseCursor (int *mouseCursorStyle);
```

Purpose

Returns the mouse cursor style you set by calling `SetMouseCursor`.

Parameters

Output

Name	Type	Description
mouseCursorStyle	integer	Returns the mouse cursor style you set by calling `SetMouseCursor`. The possible mouse cursor styles appear in Table 3-4 of Chapter 3.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetNumAxisItems

```
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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

axis	integer	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)

Output

Name	Type	Description
count	integer	Number of string/value pairs for the axis.

Return Value

Name	Type	Description
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

Returns the number of checked list items in a list control.

This function applies only to list boxes for which the check mode attribute is currently enabled.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
numberOfItems	integer	Returns the number of checked items in the list box.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetNumListItems

```
int status = GetNumListItems (int panelHandle, int controlID, int *count);
```

Purpose

Returns the number of list items in a list control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
count	integer	Returns the number of items in the list control. Because the indices are zero based, this value is one greater than the index of the last item.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetNumTextBoxLines

```
int status = GetNumTextBoxLines (int panelHandle, int controlID, int *count);
```

Purpose

Returns the number of lines of text in a text box, including lines not currently in the viewing area of the text box.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Output

Name	Type	Description
count	integer	Returns the number of lines of text in the text box.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetPanelAttribute

```
int status = GetPanelAttribute (int panelHandle, int panelAttribute,
                                void *attributeValue);
```

Purpose

Returns the value of a specific panel attribute.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
panelAttribute	integer	Refer to Table 3-2 in Chapter 3 for a complete listing of panel attributes.

Output

Name	Type	Description
attributeValue	void *	Current value of the panel attribute. Refer to Table 3-2 in Chapter 3 for a complete listing of all panel attributes, their data types, and their valid values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetPanelDisplayBitmap

```
int status = GetPanelDisplayBitmap (int panelHandle, int scope, Rect area,
                          int *bitmapID);
```

Purpose

Creates a bitmap object containing a screen shot image of the current appearance of a panel. You can pass the bitmap ID to 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 `DiscardBitmap`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
scope	integer	Specifies which portions of the panel to copy to the bitmap. Refer to the table of valid values that follows this table.
area	Rect	Restricts the area of the panel to copy 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 to copy.

Output

Name	Type	Description
bitmapID	integer	ID that serves as a handle to the bitmap object.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

The following table shows valid values for the **scope** parameter.

Valid Values	Description
`VAL_VISIBLE_AREA`	Copy the visible area of the panel to the bitmap, including the frame, menubar, and scroll bars.
`VAL_FULL_PANEL`	Copy the entire contents of the panel to the bitmap, excluding the frame, menubar, and scroll bars. This includes contents that might currently be scrolled off the visible area.

See Also

`ClipboardPutBitmap`, `GetBitmapData`, `GetCtrlDisplayBitmap`, `SetCtrlBitmap`, `PlotBitmap`, `CanvasDrawBitmap`, `DiscardBitmap`, `GetScaledPanelDisplayBitmap`

GetPanelMenuBar

```
int menuBarHandle = GetPanelMenuBar (int panelHandle);
```

Purpose

Returns the handle of the menubar associated with a specific panel.

Parameter

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Return Value

Name	Type	Description
menuBarHandle	integer	Returns the handle of the menubar associated with the panel. Returns zero if no menubar is associated with the panel. Refer to Appendix A for error codes.

GetPlotAttribute

```
int status = GetPlotAttribute (int panelHandle, int controlID,
                               int plotHandle, int plotAttribute,
                               void *attributeValue);
```

Purpose

Obtains the value of a graph plot attribute.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
plotHandle	integer	Handle for a particular plot in the graph. You obtain the handle from one of the graph plotting functions.
plotAttribute	integer	Selects a particular graph plot attribute. Refer to Tables 3-59 to 3-60 in Chapter 3 for a complete listing of plot attributes.

Output

Name	Type	Description
attributeValue	void *	Current value of the plot attribute. Refer to Tables 3-59 to 3-60 in Chapter 3 for a complete listing of plot attributes.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetPrintAttribute

```
int status = GetPrintAttribute (int printAttribute, void *attributeValue);
```

Purpose

Returns the value of a specific print attribute.

Parameter

Input

Name	Type	Description
printAttribute	integer	Refer to Tables 3-69 to 3-70 in Chapter 3 for a complete listing of print attributes.

Output

Name	Type	Description
attributeValue	void *	Current value of the print attribute. Refer to Tables 3-69 to 3-70 in Chapter 3 for a complete listing of all print attributes, their data types, and their valid values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetRelativeMouseState

```
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** return the position of the mouse relative to the top and left coordinates of a specific control. If you pass zero as the **controlID**, the coordinate information references the top and left coordinates of the panel, excluding the panel frame and title bar. `GetRelativeMouseState` returns the coordinates even if the mouse is not over the control or the panel.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`. Pass `0` to request coordinates relative to the panel.

Output

Name	Type	Description
xCoordinate	integer	X-coordinate of the mouse cursor relative to the left edge of the control, or panel, if control ID is `0`.
yCoordinate	integer	Y-coordinate of the mouse cursor relative to the top of the control, or panel, if controlID is `0`.
leftButtonDown	integer	`0` if the left button is up. `1` if the left button is down.

Name	Type	Description
rightButtonDown	integer	`0` if the right button is up. `1` if the right button is down.
keyModifiers	integer	State of the keyboard modifiers, in other words, the <Ctrl> and <Shift> 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`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You can pass `NULL` (`0`) in place of any of the output parameters.

See Also

`GetGlobalMouseState`

GetScaledCtrlDisplayBitmap

```
int status = GetScaledCtrlDisplayBitmap (int panelHandle, int controlID,
                                         int includeLabel, int newHeight, int newWidth,
                                         int *bitmapID);
```

Purpose

Creates a bitmap object that contains a screen shot image of the current appearance of a control. **newHeight** and **newWidth** determine the dimensions of the bitmap. The image stretches or shrinks to fit into the bitmap.

You can use `GetScaledCtrlDisplayBitmap` to retrieve a scaled bitmap that you can pass to `ClipboardPutBitmap`. From the clipboard, you can paste the picture of the control into another application in the size you want for that application.

To discard the bitmap object, you pass its ID to `DiscardBitmap`.

Parameter List

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel contained in memory. `LoadPanel`, `NewPanel`, or `DuplicatePanel` will have returned this handle.
controlID	integer	Defined constant located in the `.uir` header file that you assign to the control in the User Interface Editor, or the ID that `NewCtrl` or `DuplicateCtrl` returns.
includeLabel	integer	If nonzero, the control label appears in the image.
newHeight	integer	Specifies the height in pixels of the bitmap image. Values: 1 to 32,767, or pass `-1` to use the height of the control on the screen.
newWidth	integer	Specifies the width in pixels of the bitmap image. Values: 1 to 32,767, or pass `-1` to use the width of the control on the screen.

Output

Name	Type	Description
bitmapID	integer	ID that serves as a handle to the bitmap object.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`GetBitmapData`, `GetCtrlDisplayBitmap`, `GetScaledPanelDisplayBitmap`, `SetBitmapData`, `SetCtrlBitmap`, `PlotBitmap`, `CanvasDrawBitmap`, `ClipboardPutBitmap`, `DiscardBitmap`

GetScaledPanelDisplayBitmap

```
int status = GetScaledPanelDisplayBitmap (int panelHandle, int scope,
                           Rect area, int newHeight, int newWidth,
                           int *bitmapID);
```

Purpose

Creates a bitmap object that contains a screen shot image of the current appearance of a panel. **newHeight** and **newWidth** determine the dimensions of the bitmap. The image stretches or shrinks to fit into the bitmap.

You can use `GetScaledPanelDisplayBitmap` to retrieve a scaled bitmap that you can pass to `ClipboardPutBitmap`. From the clipboard, you can paste the picture of the control into another application in the size you want for that application.

To discard the bitmap object, you pass its ID to `DiscardBitmap`.

Parameter List

Name	Type	Description
panelHandle	integer	Specifier for a particular panel contained in memory. `LoadPanel`, `NewPanel`, or `DuplicatePanel` will have returned this handle.
scope	integer	Specifies which portions of the panel to copy to the bitmap. Refer to the table of valid values that follows this table.
area	Rect	Restricts the area of the panel to copy into the bitmap. The rectangle coordinates are relative to the upper-left corner of the panel, directly below the title bar. Use `VAL_ENTIRE_OBJECT` if you do not want to restrict the area to copy.
newHeight	integer	Specifies the desired height in pixels of the bitmap image. Values: 1 to 32,767, or pass `-1` to use the height of the panel on the screen. Refer to the discussion that follows.

Name	Type	Description
newWidth	integer	Specifies the desired width in pixels of the bitmap image. Values: 1 to 32,767, or pass `-1` to use the width of the panel on the screen. Refer to the discussion that follows.
bitmapID	integer	ID that serves as a handle to the bitmap object.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

The following table shows valid values for the **scope** parameter.

Valid Values	Description
`VAL_VISIBLE_AREA`	Copy the visible area of the panel to the bitmap, including the frame, menubar, and scroll bars.
`VAL_FULL_PANEL`	Copy the entire contents of the panel to the bitmap, excluding the frame, menubar, and scroll bars. This includes contents that might currently be scrolled off the visible area.

If you pass `VAL_ENTIRE_OBJECT` for the **area** parameter, **newHeight** and **newWidth** specify the exact height and width of the scaled bitmap. Otherwise, `GetScaledPanelDisplayBitmap` computes the bitmap height and width as follows:

$$\text{bitmap height} = \frac{\textbf{newHeight}}{\text{height of panel on screen}} \times \text{area.height}$$

$$\text{bitmap width} = \frac{\textbf{newWidth}}{\text{width of panel on screen}} \times \text{area.width}$$

See Also

`GetBitmapData`, `GetPanelDisplayBitmap`, `GetScaledCtrlDisplayBitmap`, `SetBitmapData`, `SetCtrlBitmap`, `PlotBitmap`, `CanvasDrawBitmap`, `ClipboardPutBitmap`, `DiscardBitmap`

GetScreenSize

```
int status = GetScreenSize (int *height, int *width);
```

Purpose

Returns the height and width of the screen in pixels.

Parameters

Output

Name	Type	Description
height	integer	Height of the screen in pixels.
width	integer	Width of the screen in pixels.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetSharedMenuBarEventPanel

```
int panelHandle = GetSharedMenuBarEventPanel (void);
```

Purpose

Returns the handle for the panel on which the last menu commit event occurred.

This function is useful when you share one menubar among multiple panels, and you use `GetUserEvent` to get menu commit events rather than using callback functions.

Return Value

Name	Type	Description
panelHandle	integer	Returns the handle for the panel on which the last commit event occurred. Refer to Appendix A for error codes.

GetSleepPolicy

```
int sleepPolicy = GetSleepPolicy (void);
```

Purpose

Obtains the current sleep policy. Refer to `SetSleepPolicy` for more information on sleeping.

Return Value

Name	Type	Description
sleepPolicy	integer	The current sleep policy.

Return Codes

Name	Code	Description
`VAL_SLEEP_NONE`	1	Never be put to sleep.
`VAL_SLEEP_SOME`	2	Be put to sleep for a short period.
`VAL_SLEEP_MORE`	3	Be put to sleep for a longer period.

See Also

`SetSleepPolicy`

GetSystemAttribute

```
int status = GetSystemAttribute (int systemAttribute, void *attributeValue);
```

Purpose

Returns the value of a particular system attribute.

Parameters

Input

Name	Type	Description
systemAttribute	integer	The particular system attribute. See Table 3-66 in Chapter 3 for a complete listing of system attributes.

Output

Name	Type	Description
attributeValue	void *	Current value of the system attribute. Refer to Table 3-66 in Chapter 3 for a complete listing of system attributes, their data types, and their valid values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetSystemPopupsAttribute

```
int status = GetSystemPopupsAttribute (int popupAttribute,
                           void *attributeValue);
```

Purpose

Returns the value of a specific system pop-up attribute.

The system popup attributes apply to all of the LabWindows/CVI popup panels, such as the graph pop-ups, the message pop-ups, and the file select popup on Windows 3.1 and UNIX.

The system popup attributes do not apply to the dialog boxes that are native to the Windows operating systems, such as the Windows 95/NT file dialog box. `FileSelectPopup`, `MultiFileSelectPopup`, and `DirSelectPopup` use the Windows 95/NT file dialog box.

Parameters

Input

Name	Type	Description
popupAttribute	integer	`ATTR_MOVABLE` or `ATTR_SYSTEM_MENU_VISIBLE`

Output

Name	Type	Description
attributeValue	void*	Current value of the specified attribute. Refer to Table 3-2 in Chapter 3 for valid values associated with these attributes.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetTextBoxLine

```
int status = GetTextBoxLine (int panelHandle, int controlID, int lineIndex,
                             char destinationBuffer[]);
```

Purpose

Returns a string containing the text from a specific text box line.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
lineIndex	integer	Zero-based index into the text box.

Output

Name	Type	Description
destinationBuffer	string	Returns a string that contains the text from the text box line.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`GetTextBoxLineLength`

GetTextBoxLineIndexFromOffset

```
int status = GetTextBoxLineIndexFromOffset (int panelHandle, int controlID,
                                    int byteOffset, int *lineIndex)
```

Purpose

For the text box control you specify, returns the zero-based index of the line that contains the character at **byteOffset** bytes beyond the beginning of the text.

Parameter List

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
byteOffset	integer	Number of bytes beyond the beginning of the text currently contained in the text box, including any embedded newline characters. Must be greater than or equal to zero.

Output

Name	Type	Description
lineIndex	integer	Returns the zero-based index of the line that contains the character that is at **byteOffset** bytes beyond the beginning of the text. If **byteOffset** exceeds the number of bytes in the text, returns the number of lines in the text box.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`GetTextBoxLineOffset`, `GetTextBoxLineLength`

GetTextBoxLineLength

```
int status = GetTextBoxLineLength (int panelHandle, int controlID,
                                   int lineIndex, int *length);
```

Purpose

Returns the number of characters in a specific text box line, including ASCII NUL characters.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
lineIndex	integer	Zero-based index into the text box.

Output

Name	Type	Description
length	integer	Returns number of characters in the text box line.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`GetTextBoxLineOffset`, `GetTextBoxLine`

GetTextBoxLineOffset

```
int status = GetTextBoxLineOffset (int panelHandle, int controlID,
                                   int lineIndex, int *byteOffset)
```

Purpose

Returns the byte offset of a specific line in a text box, from the beginning of the text.

Parameter List

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
lineIndex	integer	Zero-based index of the line in the text box.

Output

Name	Type	Description
byteOffset	integer	Returns the number of bytes in the text box before the beginning of the line.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`GetTextBoxLineIndexFromOffset`, `GetTextBoxLineLength`

GetTextDisplaySize

```
int status = GetTextDisplaySize (char text[], char metaFont[], int *height,
                                 int *width);
```

Purpose

Returns the height and width of a string of text given the metafont you specify.

Parameters

Input

Name	Type	Description
text	string	Text string whose size is to be calculated.
metaFont	string	Metafont to use in calculating the display size of the text.

Output

Name	Type	Description
height	integer	Height of the text in pixels.
width	integer	Width of the text in pixels.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetTraceAttribute

```
int status = GetTraceAttribute (int panelHandle, int controlID,
                               int traceNumber, int traceAttribute,
                               int *attributeValue);
```

Purpose

Obtains the value of a strip chart trace attribute.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
traceNumber	integer	Identifies a strip chart trace. Can be from `1` to the number of strip chart traces. You set the number of strip chart traces in the User Interface Editor or through `SetCtrlAttribute`.
traceAttribute	integer	Selects a particular strip chart trace attribute. Refer to Table 3-59 in Chapter 3 for a complete listing of trace attributes.

Output

Name	Type	Description
attributeValue	integer	Current value of the trace attribute.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetUILErrorString

```
char *message = GetUILErrorString (int errorNum)
```

Purpose

Converts the error number returned by a User Interface Library function into a meaningful error message.

Parameter

Input

Name	Type	Description
errorNum	integer	Status returned by User Interface Library function.

Return Value

Name	Type	Description
message	string	Explanation of error.

GetUserEvent

```
int event = 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. You place programmer-defined events in the `GetUserEvent` queue by calling `QueueUserEvent`.

Parameters

Input

Name	Type	Description
waitMode	integer	If `1`, `GetUserEvent` does not return until a commit event or programmer-defined event occurs. If `0`, `GetUserEvent` returns immediately, whether or not a commit event or programmer-defined event has occurred.

Output

Name	Type	Description
panelOrMenuBarHandle	integer	Returns the handle of the panel or menubar on which the event occurred. Returns `-1` if **waitMode** is zero and no event has occurred.
controlOrMenuItemID	integer	Returns the ID of the control or menu item on which the commit event occurred. Returns `-1` if **waitMode** is zero and no event has occurred.

Return Value

Name	Type	Description
event	integer	The event, if any, that occurred. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

Return Codes

Contains the event retrieved from the `GetUserEvent` queue. The following values are possible:

Code	Description
`0`	No event.
`1`	Commit event occurred.
`1000-10000`	Event you queued by calling `QueueUserEvent`.

GetValueFromIndex

```
int status = GetValueFromIndex (int panelHandle, int controlID,
                                int itemIndex, void *itemValue);
```

Purpose

Returns the value of a specific list item.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list.

Output

Name	Type	Description
itemValue	void *	Returns the value of the list item you specify.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

GetValueLengthFromIndex

```
int status = GetValueLengthFromIndex (int panelHandle, int controlID,
                                      int itemIndex, int *length);
```

Purpose

Returns the length of the value of a specific list item.

This function is valid only for list controls with string data type.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list.

Output

Name	Type	Description
length	integer	Returns the length of the value of the list item you specify.

Return Value

Name	Type	Description
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

Name	Type	Description
cursorState	integer	The state of the wait cursor. Negative values indicate an error occurred. Refer to Appendix A for error codes.

Return Codes

Code	Description
`0`	Wait cursor inactive.
`1`	Wait cursor active.

HidePanel

```
int status = HidePanel (int panelHandle);
```

Purpose

Clears a panel from the screen but leaves it in memory.

After you hide a panel, you can still update it programmatically. For example, you can add plots to a graph control on a hidden panel. When you display the panel again, it shows the new plots.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

InsertAxisItem

```
int status = InsertAxisItem (int panelHandle, int controlID, int axis,
                             int itemIndex, char itemLabel[],
                             double itemValue);
```

Purpose

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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

axis	integer	Specifies the axis for which to insert the string/value pair. Valid values:
		`VAL_XAXIS`
		`VAL_LEFT_YAXIS`
		`VAL_RIGHT_YAXIS` (graphs only)

itemIndex	integer	Zero-based index into the list at which to store the item. Pass `-1` to store the item at the end of the list. Refer to discussion below.

Name	Type	Description
itemLabel	string	Label string to insert. The label appears on the axis at the location of the value you specify. An axis label displays a maximum of 31 characters.
itemValue	double-precision	Value with which to associate the label string. The label string appears on the axis at the location of the value.

Return Value

Name	Type	Description
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

```
int status = InsertListItem (int panelHandle, int controlID, int itemIndex,
                             char itemLabel[], ...);
```

Purpose

Inserts a label/value pair into a list or ring control at a specific zero-based index.

Inserting the new pair causes the indices of existing label/value pairs at and beyond the insertion point to increase by one.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list at which to place the item. Pass `-1` to insert the item at the end of the list.
itemLabel	string	Label to associate with **itemValue**.
itemValue	depends on type of list control	Value to associate with **itemLabel**. The data type must be the same as the data type of the control.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

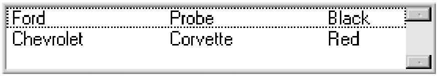
Parameter Discussion

For picture rings, the *label* is actually an image, and you pass the pathname of the image as the `itemLabel` parameter. The image pathname can be a complete pathname or a simple filename. If a simple filename, the image file must be in the project or in the directory of the project. If you pass `NULL` or the empty string, LabWindows/CVI creates a placeholder for the image that you can fill 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);
```

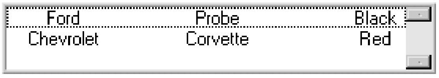
The preceding code segment creates the following tab format:



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:



To insert a vertical line at the current position in the label, the code is `\033vline`.

You can also use escape codes to change the foreground and background colors of characters in a list box. The escape codes affect only the label in which they are embedded. The new color affects subsequent characters in the label until you change the color again.

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 restore either color 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 Popup Menu Ring) by embedding the escape code `\033m-` in the `itemLabel` string.

See Also

`ReplaceListItem, DeleteListItem, ClearListCtrl`

InsertSeparator

```
int menuItemID = InsertSeparator (int menuBarHandle, int menuID,
                                  int beforeMenuItemID);
```

Purpose

Inserts a new separator bar in a menu, and returns a menu item ID you can use in subsequent function calls to specify the separator.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.
menuID	integer	ID for a particular menu within a menubar. The Menu ID should be a constant name (located in the `.uir` header file) that the User Interface Editor generates or a value you obtain from `NewMenu`.
beforeMenuItemID	integer	ID of the menu item above which to insert the separator bar. To place the separator at the bottom of the menu item list, pass `-1`.

Return Value

Name	Type	Description
menuItemID	integer	ID you use to reference the separator in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

InsertTextBoxLine

```
int status = InsertTextBoxLine (int panelHandle, int controlID,
                                int lineIndex, char text[]);
```

Purpose

Inserts a string as a new line in a text box.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
lineIndex	integer	Zero-based index of the text box line above which to insert the new line. Pass `-1` to insert the new line at the end of the text box.
text	integer	Text to insert as a new line.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

A text box can have only a limited number of lines. The number of lines multiplied by the pixel height of the font must be less than 32,767, or else the text box does not scroll. In a text box that uses `NIDialogMetaFont` or `NIEditorMetaFont`, you can have a maximum of about 2,500 lines.

InstallCtrlCallback

```
int status = InstallCtrlCallback (int panelHandle, int controlID,
                                  CtrlCallbackPtr eventFunction,
                                  void *callbackData);
```

Purpose

Installs a control callback.

After you install the callback, it is called whenever the user generates an event on the control.

The event function (type `CtrlCallbackPtr`) takes the following form:

```
int CVICALLBACK EventFunctionName (int panelHandle, int controlID,
                                   int event, void *callbackData,
                                   int eventData1, int eventData2);
```

The event function receives the panel handle and control ID of the control generating the event. It also receives 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 appear 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.

The **callbackData** you pass to `InstallCtrlCallback` is passed to the callback function. You can use the **callbackData** to hold a pointer to your own information and thereby avoid the use of global variables.

You do not have to call `InstallCtrlCallback` if you already associated a callback function with the control in the User Interface Editor.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Name	Type	Description
eventFunction	CtrlCallbackPtr	Name of the function that processes the control events.
callbackData	void *	Points to data that you define. The callback function receives the pointer.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

InstallMainCallback

```
int status = InstallMainCallback (MainCallbackPtr eventFunction,
                                  void *callbackData, int getIdleEvents);
```

Purpose

Installs the main callback. For a discussion of the main callback, refer to the *Special User Interface Functions* section in Chapter 3, *Programming with the User Interface Library.*

After you install the callback, the event function is called whenever an event is generated and not swallowed by another callback.

The event function (type `MainCallbackPtr`) takes the following form:

```
int CVICALLBACK EventFunctionName (int panelOrMenuBarHandle,
                                   int controlOrMenuItemID, int event,
                                   void *callbackData, int eventData1,
                                   int eventData2);
```

The event function receives information on the source of the callback, either the panel handle and control ID, or the menubar handle and menu item ID. It also receives 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

Input

Name	Type	Description
eventFunction	MainCallbackPtr	Name of the function that processes events.
callbackData	void *	Points to data that you define. The callback function receives the pointer.
getIdleEvents	integer	`1` = respond to idle events `0` = ignore idle events

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

If you pass `1` for **getIdleEvents**, the callback receives idle events regularly at the rate you specify with `SetIdleEventRate`, as long as you allow LabWindows/CVI to process events. You can achieve the same effect by using a timer control. National Instruments recommends that you use timer controls rather than a main callback with idle events.

InstallMenuCallback

```
int status = InstallMenuCallback (int menuBarHandle, int menuOrMenuItemID,
                          MenuCallbackPtr eventFunction,
                          void *callbackData);
```

Purpose

Installs a menu callback for a specific menu or menu item.

After you install the callback, it is called whenever the user generates an event on the menu or menu item.

The event function (type `MenuCallbackPtr`) takes the following form:

```
void CVICALLBACK EventFunctionName (int menuBarHandle,
                          int menuItemID, void *callbackPtr,
                          int panelHandle);
```

When a commit event is generated by a menu selection, the event function receives the menubar handle, menu item ID, and panel handle of the menubar generating the event. The **callbackData** you pass to `InstallMenuCallback` is passed to the event function.

You do not have to call `InstallMenuCallback` if you already associated a callback function with the menubar through one of the following mechanisms: the Edit Menu Bar dialog box in the User Interface Editor or by installing it through the `NewMenuItem` function.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar.`
menuOrMenuItemID	integer	Menu or menu item ID assigned in the User Interface Editor or you obtain from the `NewMenu` or `NewMenuItem.`
eventFunction	MenuCallbackPtr	Name of the function that processes the menu events.
callbackData	void *	Points to data that you define. The callback function receives the pointer.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

InstallMenuDimmerCallback

```
int status = InstallMenuDimmerCallback (int menuBarHandle,
                             MenuDimmerCallbackPtr dimmerFunction);
```

Purpose

Installs a menu dimmer callback for a specific menubar.

LabWindows/CVI invokes your menu dimmer callback after the user clicks on a menu name on the menubar and before the menu appears. This give you the opportunity to dim menu items based on your program's current state. LabWindows/CVI also invokes your menu dimmer callback when the user presses a shortcut key for any menu item in the menubar. If your menu dimmer callback dims the menu item or the item is already dim, LabWindows/CVI does not generate a commit event for the menu item.

The callback function (type `MenuDimmerCallbackPtr`) takes the following form:

```
void CVICALLBACK DimmerFunctionName (int menuBarHandle,
                                     int panelHandle);
```

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.
dimmerFunction	MenuDimmerCallbackPtr	Name of the menu dimmer callback function.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

InstallPanelCallback

```
int status = InstallPanelCallback (int panelHandle,
                                   PanelCallbackPtr eventFunction,
                                   void *callbackData);
```

Purpose

This function installs a panel callback.

After you install the callback, it is called whenever the user generates an event on the panel.

The event function (type `PanelCallbackPtr`) takes the following form:

```
int CVICALLBACK EventFunctionName (int panelHandle, int event,
                                   void *callbackData, int eventData1,
                                   int eventData2);
```

The event function receives the panel handle of the panel generating the event, 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. Refer to Table 3-1 of 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.

The **callbackData** you pass to `InstallPanelCallback` is passed to the callback function. You can use **callbackData** to hold a pointer to your own information and thereby avoid the use of global variables.

You do not have to call `InstallPanelCallback` if you already associated a callback function with the panel in the User Interface Editor.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
eventFunction	PanelCallbackPtr	Name of the function that processes the panel events.
callbackData	void *	Points to data that you define. The callback function receives the pointer.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

InstallPopup

```
int status = InstallPopup (int panelHandle);
```

Purpose

Displays and activates a panel as a dialog box.

You must call this function from the thread in which you create the panel.

While the modal dialog box is visible, the user cannot operate any other panels that you created in the same thread.

Parameter

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`RemovePopup`

IsListItemChecked

```
int status = IsListItemChecked (int panelHandle, int controlID,
                                int itemIndex, int *checked);
```

Purpose

Returns a Boolean value indicating whether or not a specific list item is checked.

This function only applies to list boxes for which the check mode attribute is enabled.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list control.

Output

Name	Type	Description
checked	integer	Returns a Boolean value indicating whether the list item is checked. `0` = not checked `1` = checked

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

LoadMenuBar

```
int menuBarHandle = LoadMenuBar (int destinationPanelHandle,
                           char filename[], int menuBarResourceID);
```

Purpose

Loads a menubar into memory from a user interface resource (`.uir`) file or text user interface (`.tui`) file you created in the User Interface Editor. The menubar resides on the panel you specify by the destination panel handle.

The function returns a menubar handle that you use in subsequent function calls to specify the menubar.

Note ***The*** `ATTR_REPORT_LOAD_FAILURE` ***and*** `ATTR_ALLOW_MISSING_CALLBACKS` ***system attributes affect how this function behaves when it encounters an error. Refer to the*** *Using the System Attributes* ***discussion in Chapter 3,*** *Programming with the User Interface Library***.**

Parameters

Input

Name	Type	Description
destinationPanelHandle	integer	Handle of the panel on which to place the menubar.
filename	string	Name of the user interface resource file that contains the menu bar.
menuBarResourceID	integer	Defined constant that you assigned to the menubar in the User Interface Editor.

Return Value

Name	Type	Description
menuBarHandle	integer	Value you can use in subsequent function calls to specify this menubar. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

Parameter Discussion

You obtain **destinationPanelHandle** from `LoadPanel`, `NewPanel`, or `DuplicatePanel`. If the destination panel is not currently in memory, pass a zero as the destination panel handle. You can later assign the menubar 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 that contains no directory path, and it is listed in the project, then the file is loaded from the project. Otherwise, the file is loaded from the directory containing the project.

The **menuBarResourceID** is in the `.uir` header file, and you use it only to load the menubar into memory. You use the menubar handle this function returns to refer to the menubar in subsequent function calls.

Details on Loading Menu Bars from .tui Files

When you load a menubar from a text user interface (`.tui`) file, the **menuBarResourceID** parameter must be the header number of the `.tui` file section that defines the panel. For example, if the section header for the menubar you want is `[MenuBar003]`, pass `3` as the Menu Bar Resource ID.

This function loads all of the menus and menu items in the `.tui` file whose section headers take the form `[MenuBar`*`NNN`*`_...]`, where *`NNN`* is the **menuBarIDResourceID**. The menu ID or menu item ID that you pass to User Interface Library functions is based on a depth-first traversal of all of the items in the menu tree, starting at `2`. For submenu items, the submenu has an ID that is one greater than the item ID of the submenu entry in the parent menu.

If you save a `.tui` file in the User Interface Editor in LabWindows/CVI 5.0 or later, and you have an up-to-date include file (`.h`) that the User Interface Editor generated, you can use the menubar, menu, submenu, and menu item constants in the include file as parameters to User Interface Library functions.

See Also

`LoadMenuBarEx`, `SetSystemAttribute`

LoadMenuBarEx

```
int menuBarHandle = LoadMenuBarEx (int destinationPanelHandle, filename[],
                           int menuBarResourceID,
                           void *callingModuleHandle);
```

Purpose

`LoadMenuBarEx` loads a menubar into memory from a user interface resource (`.uir`) file or text user interface (`.tui`) file you created in the User Interface Editor. `LoadMenuBarEx` is similar to `LoadMenuBar`, except that, when you use `LoadMenuBarEx` on Windows 95/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/NT, `LoadMenuBarEx` works exactly like `LoadMenuBar`.

Note ***The*** `ATTR_REPORT_LOAD_FAILURE` ***and*** `ATTR_ALLOW_MISSING_CALLBACKS` ***system attributes affect how this function behaves when it encounters an error. Refer to the*** *Using the System Attributes* ***discussion in Chapter 3,*** *Programming with the User Interface Library****.***

Parameters

Input

Name	Type	Description
destinationPanelHandle	integer	Handle of the panel on which to place the menubar.
filename	string	Name of the user interface resource file that contains the menu bar.
panelResourceID	integer	Defined constant you assigned to the menubar in the User Interface Editor.
callingModuleHandle	void pointer	Usually, the module handle of the calling DLL. You can use `__CVIUserHInst`. Zero indicates the project or executable.

Return Value

Name	Type	Description
menuBarHandle	integer	Value you can use in subsequent function calls to specify this menubar. 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 you define in the project or in object, library, or DLL import library modules that you have already loaded using `LoadExternalModule`. It does not search symbols that you define in, but do not export from, a DLL.

If you want to load a menubar in a DLL that defines, but does not export, the menu callback functions, use `LoadMenuBarEx`. 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 `LoadMenuBar`.

`LoadMenuBarEx` first searches the DLL symbols to find the callback functions referenced in the `.uir`. If any callback functions exist 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 the DLL loads. If you create the DLL in LabWindows/CVI, LabWindows/CVI creates the table for you 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 the `.uir` files the DLL loads. 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.

Details on Loading Menu Bars from .tui Files

When you load a menubar from a text user interface (`.tui`) file, the **menuBarResourceID** parameter must be the header number of the `.tui` file section that defines the panel. For example, if the section header for the desired menubar is `[MenuBar003]`, pass `3` as the Menu Bar Resource ID.

This function loads all of the menus and menu items in the `.tui` file whose section headers take the form `[MenuBar`*`NNN`*`_...]`, where *`NNN`* is the **menuBarIDResourceID**. The menu ID or menu item ID that you pass to User Interface Library functions is based on a depth-first traversal of all of the items in the menu tree, starting at `2`. For submenu items, the submenu has an ID that is one greater than the item ID of the submenu entry in the parent menu.

If you save a `.tui` file in the User Interface Editor in LabWindows/CVI 5.0 or later, and you have an up-to-date include file (`.h`) that the User Interface Editor generated, you can use the menubar, menu, submenu, and menu item constants in the include file as parameters to User Interface Library functions.

See Also

`LoadMenuBar`, `LoadPanelEx`, `SetSystemAttribute`

LoadPanel

```
int panelHandle = LoadPanel (int parentPanelHandle, char filename[],
                             int panelResourceID);
```

Purpose

Loads a panel into memory from a user interface resource (`.uir`) file or text user interface (`.tui`) file you created in the User Interface Editor.

The panel becomes a child panel of the parent panel you specify by **parentPanelHandle**. To make the panel a top-level panel, pass `0` for the **parentPanelHandle**.

The function returns a panel handle that you use in subsequent function calls to specify the panel. You must call `DisplayPanel` to make the panel visible.

If you call this function to load a panel as a child panel, you must do so in the same thread in which you create the parent panel.

Note ***The*** `ATTR_REPORT_LOAD_FAILURE` ***and*** `ATTR_ALLOW_MISSING_CALLBACKS` ***system attributes affect how this function behaves when it encounters an error. Refer to the*** *Using the System Attributes* ***discussion in Chapter 3,*** *Programming with the User Interface Library***.**

Parameters

Input

Name	Type	Description
parentPanelHandle	integer	Handle of the panel into which to load the panel as a child panel. Pass `0` to load the panel as a top-level window.
filename	string	Name of the user interface resource file that contains the panel.
panelResourceID	integer	Defined constant that you assigned to the panel in the User Interface Editor.

Return Value

Name	Type	Description
panelHandle	integer	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**. To load the panel 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 that contains no directory path, and it is listed in the project, then the file is loaded from the project. Otherwise, the file is loaded from the directory containing the project.

The **panelResourceID** is found in the `.uir` header file and you use it only to load the panel into memory. You use the panel handle this function returns to refer to the panel in subsequent function calls.

Details on Loading Panels from .tui Files

When you load a panel from a text user interface (`.tui`) file, the **panelResourceID** parameter must be the header number of the `.tui` file section that defines the panel. For example, if the section header for the desired panel is `[Panel003]`, pass `3` as the **panelResourceID**.

This function loads all of the controls in the `.tui` file whose section headers take the form `[Panel`*`NNN`*`_Control`*`YYY`*`]`, where *`NNN`* is the **panelResourceID** and *`YYY`* is `001` or greater. The control numbers must be consecutive and start at `001`. To specify the control in calls to other User Interface Library functions, pass *`YYY`* + `1` as the **controlID**. For example, if the section header for the control is `[Panel003_Control001]`, pass `2` as the **controlID** parameter.

If you save a `.tui` file in the User Interface Editor in LabWindows/CVI 5.0 or later, and you have an up-to-date include file (`.h`) that the User Interface Editor generated, you can use the panel and control constants in the include file as parameters to User Interface Library functions.

See Also

`LoadPanelEx`, `SetSystemAttribute`

LoadPanelEx

```
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 or text user interface (`.tui`) file created in the User Interface Editor. `LoadPanelEx` is similar to `LoadPanel`, except that, when you use `LoadPanelEx` your program on Windows 95/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/NT, `LoadPanelEx` works exactly like `LoadPanel`.

Note ***The*** `ATTR_REPORT_LOAD_FAILURE` ***and*** `ATTR_ALLOW_MISSING_CALLBACKS` ***system attributes affect how this function behaves when it encounters an error. Refer to the*** *Using the System Attributes* ***discussion in Chapter 3,*** *Programming with the User Interface Library****.***

Parameters

Input

Name	Type	Description
parentPanelHandle	integer	Handle of the panel into which to load the panel as a child panel. Pass `0` to load the panel as a top-level window.
filename	string	Name of the user interface resource file that contains the panel.
panelResourceID	integer	Defined constant that you assigned to the panel in the User Interface Editor.
callingModuleHandle	void pointer	Usually, the module handle of the calling DLL. You can use `_CVIUserHInst`. Zero indicates the project or executable.

Return Value

Name	Type	Description
panelHandle	integer	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 you define in the project or in object, library, or DLL import library modules you have already loaded using `LoadExternalModule`. It does not search symbols that you define in, but do not export from, a DLL.

If you want to load a panel in a DLL that defines, but does not export, the panel callback functions, use `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 any callback functions exist 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 the DLL loads. If you create the DLL in LabWindows/CVI, LabWindows/CVI creates the table 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 the `.uir` files the DLL loads. 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.

Details on Loading Panels from .tui Files

When you load a panel from a text user interface (`.tui`) file, the **panelResourceID** parameter must be the header number of the `.tui` file section that defines the panel. For example, if the section header for the desired panel is `[Panel003]`, pass `3` as the **panelResourceID**.

This function loads all of the controls in the `.tui` file whose section headers take the form `[Panel`*`NNN`*`_Control`*`YYY`*`]`, where *`NNN`* is the **panelResourceID** and `YYY` is `001` or greater. The control numbers must be consecutive and start at `001`. To specify the control in calls to other User Interface Library functions, pass *`YYY`* `+ 1` as the **controlID**. For example, if the section header for the control is `[Panel003_Control001]`, pass `2` as the **controlID** parameter.

If you save a `.tui` file in the User Interface Editor in LabWindows/CVI 5.0 or later, and you have an up-to-date include file (`.h`) that the User Interface Editor generated, you can use the panel and control constants in the include file as parameters to User Interface Library functions.

See Also

`LoadPanel`, `LoadMenuBarEx`, `SetSystemAttribute`

MakeApplicationActive

```
int status = MakeApplicationActive (void);
```

Purpose

Under Windows, `MakeApplicationActive` activates your application and brings its topmost panel to the front. `MakeApplicationActive` has no effect if you have not displayed any panels.

Under UNIX, `MakeApplicationActive` has no effect.

Parameter List

None

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

MakeColor

```
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
0xFF00FFL = VAL_MAGENTA
0xFFFF00L = VAL_YELLOW
0x800000L = VAL_DK_RED
0x000080L = VAL_DK_BLUE
0x008000L = VAL_DK_GREEN
0x008080L = VAL_DK_CYAN
0x800080L = VAL_DK_MAGENTA
0x808000L = VAL_DK_YELLOW
0xCCCCCCL = VAL_LT_GRAY
0x808080L = VAL_DK_GRAY
0x000000L = VAL_BLACK
0xFFFFFFL = VAL_WHITE
0xA0A0A0L = VAL_GRAY
0xE5E5E5L = VAL_OFFWHITE
VAL_PANEL_GRAY = VAL_LT_GRAY
```

Parameters

Input

Name	Type	Description
red	integer	Red level of the new color. Any integer value between 0 and 255 is valid. `0` = No red component `255` = Maximum red component
green	integer	Green level of the new color. Any integer value between 0 and 255 is valid. `0` = No green component `255` = Maximum green component
blue	integer	Blue level of the new color. Any integer value between 0 and 255 is valid. `0` = No blue component `255` = Maximum blue component

Return Value

Name	Type	Description
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 values you specify. The `Point` structure defines the location of a point.

This function is useful when you call 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

Name	Type	Description
xCoordinate	integer	Horizontal location of the point.
yCoordinate	integer	Vertical location of the point.

Return Value

Name	Type	Description
point	Point	`Point` structure containing the coordinate values you specify.

See Also

`PointSet`, `MakeRect`

MakeRect

```
Rect rect = MakeRect (int top, int left, int height, int width);
```

Purpose

Returns a `Rect` structure with the values you specify. The `Rect` structure defines the location and size of a rectangle.

This function is useful when you call 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

Input

Name	Type	Description
top	integer	Location of the top edge of the rectangle.
left	integer	Location of the left edge of the rectangle.
height	integer	Height of the rectangle.
width	integer	Width of the rectangle.

Return Value

Name	Type	Description
rect	Rect	`Rect` structure containing the coordinate values you specify.

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

Name	Type	Description
title	string	Title to appear on the dialog box.
message	string	Message to display in the dialog box. To display a multi-line message, embed newline characters (`\n`) in the message string.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

MinimizeAllWindows

```
void MinimizeAllWindows (void);
```

☞ **Note** ***Only the Windows 95/NT version of LabWindows/CVI supports*** `MinimizeAllWindows`**.**

Purpose

Hides all the panels that you created in the current thread, other than panels that have their own taskbar buttons. To restore the panels, click on the taskbar button for the thread.

☞ **Note** ***If you enable the*** `ATTR_HAS_TASKBAR_BUTTON` ***attribute on your panels,*** `MinimizeAllWindows` ***hides all the taskbar buttons for the panels. To minimize each panel to its own taskbar button, set the*** `ATTR_WINDOW_ZOOM` ***attribute to*** `VAL_MINMIZE` ***on each panel.***

Parameter List

None

Return Value

None

MultiFileSelectPopup

```
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

Input

Name	Type	Description
defaultDirectory	string	Initial directory. If `""` is entered, the current working directory is used.
defaultFileSpec	string	String that specifies which files to display. For example, `"*.c"` causes all files with the extension `.c` to be displayed.
fileTypeList	string	List of file types, separated by semicolons, to display in the File Type List of the File Select Pop-up when **restrictExtension** is FALSE. For example, `"*.c;*.h"` allows the user to select `"*.c"` or `"*.h"` from the File Type List. The All Files, `*.*`, option is always available.
title	string	Title of the dialog box.
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.

Output

Name	Type	Description
numberOfSelectedFiles	integer	Number of files selected by the user.
fileList	array of strings	Buffer that contains the filenames selected by the user. The buffer is automatically allocated by `MultiFileSelectPopup` and is accessible as an array of strings.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Return Codes

Code	Description
`0`	`VAL_NO_FILE_SELECTED`
`1`	`VAL_EXISTING_FILE_SELECTED`

Parameter Discussion

The **defaultFileSpec** appears in the file name control when you initially display the pop-up. 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.

`MultiFileSelectPopup` dynamically allocates **fileList** and the strings it contains. When you no longer need them, free each string and the array with the `free` function.

NewBitmap

```
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 and returns a bitmap ID. You can pass the bitmap ID to any function that accepts a bitmap, such as `CanvasDrawBitmap` or `ClipboardPutBitmap`.

You can discard the bitmap object by passing its ID to `DiscardBitmap`.

Parameters

Input

Name	Type	Description
bytesPerRow	integer	Number of bytes on each scan line of the image.
pixelDepth	integer	Number of bits per pixel.
width	integer	Width of the image, in pixels.
height	integer	Height of the image, in pixels.
colorTable	integer array	Array of RGB color values, or `NULL` if **pixelDepth** is greater than eight.
bits	unsigned char array	Array of bits that determine the colors to display on each pixel in the image.
mask	unsigned char array	Array containing one bit per pixel in the image. Each bit specifies whether to actually draw the pixel. Can be `NULL`.

Output

Name	Type	Description
bitmapID	integer	ID that serves as a handle to the bitmap object.

Return Value

Name	Type	Description
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 eight. 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 you might have to use padding. 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`, no padding occurs 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 eight or less, the number of entries in the **colorTable** array must equal two raised to the power of the **pixelDepth** parameter. The **bits** array contain indices into the **colorTable** array. If the **pixelDepth** is greater than eight, 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 is always in the least significant byte. On little-endian platforms, for example, Intel processors, `BB` is at the lowest memory address. On big-endian platforms, for example, Motorola processors, `BB` is at the highest address. Notice that this byte ordering scheme differs from the byte ordering scheme 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. An exception applies when the **pixelDepth** is `1`. In this case, the pixels that have a **bits** value of `1`, called foreground pixels, are always drawn, and the **mask** affects only the pixels that have a value of `0`, called background pixels.

You must pad each row of the **mask** to the nearest even-byte boundary. For example, if the width of the image is 21 pixels, then each row of the mask must have 32 bits, in other words, four bytes, of data.

A mask is useful for achieving transparency.

You can pass `NULL` if you do not want a mask.

See Also

`GetBitmapFromFile`, `GetCtrlBitmap`, `GetCtrlDisplayBitmap`, `GetPanelDisplayBitmap`, `ClipboardGetBitmap`, `ClipboardPutBitmap`, `GetBitmapData`, `SetCtrlBitmap`, `PlotBitmap`, `CanvasDrawBitmap`, `DiscardBitmap`

NewCtrl

```
int controlID = NewCtrl (int panelHandle, int controlStyle,
                         char controlLabel[], int controlTop,
                         int controlLeft);
```

Purpose

Creates a new control and returns a control ID you use to specify the control in subsequent function calls.

If you call this function to add a timer control to a panel, you must do so in the same thread in which you create or load the panel.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlStyle	integer	Refer to Table 3-45 in Chapter 3 for a complete listing of all control styles.
controlLabel	string	Label of the new control. Pass `""` or `0` for no label.
controlTop	integer	Vertical coordinate at which to place the upper left corner of the control, not including labels.
controlLeft	integer	Horizontal coordinate at which to place the upper left corner of the control, not including labels.

Return Value

Name	Type	Description
controlID	integer	Returns the ID you use to specify the control in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

Parameter Discussion

The **controlTop** and **controlLeft** coordinates must be integer values from –32,768 to 32,767. The origin (0,0) is at the upper-left corner of the panel, directly below the title bar, before the panel is scrolled.

NewMenu

```
int menuID = NewMenu (int menuBarHandle, char menuName[], int beforeMenuID);
```

Purpose

Adds a new menu to a menubar and returns a menu ID you can use to specify the menu in subsequent function calls.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.
menuName	string	New menu name to add to the menubar.
beforeMenuID	integer	ID of the menu above which to insert the new menu. To place the new menu at the end of the menubar, pass `-1`.

Return Value

Name	Type	Description
status	integer	ID you use to reference the menu in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

NewMenuBar

```
int menuBarHandle = NewMenuBar (int destinationPanelHandle);
```

Purpose

Creates a new menubar to reside on a specific panel and returns a handle to the new menubar.

You use the new menubar handle in subsequent function calls to specify the menubar.

Parameter

Input

Name	Type	Description
destinationPanelHandle	integer	Handle for the panel on which to place the menubar.

Return Value

Name	Type	Description
menuBarHandle	integer	Handle you use in subsequent function calls to specify the menubar. Refer to Appendix A for error codes.

Parameter Discussion

If the destination panel is not currently in memory, pass a zero as the **destinationPanelHandle**. You can later assign the menubar to a panel using `SetPanelMenuBar`**.**

NewMenuItem

```
int menuItemID = NewMenuItem (int menuBarHandle, int menuID, char itemName[],
                              int beforeMenuItemID, int shortCutKey,
                              MenuCallbackPtr eventFunction,
                              void *callbackData);
```

Purpose

Adds a new menu item to a specific menu and returns a menu item ID you use in subsequent calls to specify the menu item.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`
menuID	integer	ID for a particular menu within a menubar. The Menu ID should be a constant name, located in the `.uir` header file, that the User Interface Editor generates or a value you obtain from `NewMenu`.
itemName	string	Name of the new menu item.
beforeMenuItemID	integer	ID of the menu item above which to insert the new item. To place the new menu item at the bottom of the menu item list, pass `-1`.
shortCutKey	integer	Specifies the key or key combination the user can press to automatically select the menu item.
eventFunction	MenuCallbackPtr	Name of the user function that process the menu item callback.
callbackData	void *	Points to data that you define. The callback function receives the pointer.

Return Value

Name	Type	Description
menuItemID	integer	Returns the ID that you use 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

This section contains lists of the 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 you construct a shortcut key, you can bitwise OR modifier keys with a virtual key or with an ASCII key. Either the ASCII field or the virtual key field must be all zeros. For example, the following construct produces a shortcut key of <Shift-F1>:

```
VAL_SHIFT_MODIFIER | VAL_F1_VKEY
```

Similarly, the following construct produces a shortcut key of <Ctrl-D>:

```
VAL_MENUKEY_MODIFIER | 'D'
```

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

The following modifier keys are available:

`VAL_SHIFT_MODIFIER`
`VAL_MENUKEY_MODIFIER` (the <Ctrl> key)
`VAL_SHIFT_AND_MENUKEY`

The following virtual keys are available:

`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`

```
VAL_RIGHT_ARROW_VKEY
VAL_INSERT_VKEY
VAL_HOME_VKEY
VAL_END_VKEY
VAL_PAGE_UP_VKEY
VAL_PAGE_DOWN_VKEY
VAL_F1_VKEY
VAL_F2_VKEY
VAL_F3_VKEY
VAL_F4_VKEY
VAL_F5_VKEY
VAL_F6_VKEY
VAL_F7_VKEY
VAL_F8_VKEY
VAL_F9_VKEY
VAL_F10_VKEY
VAL_F11_VKEY
VAL_F12_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 the **eventFunction** parameter. The event function, type `MenuCallbackPtr`, is prototyped as follows:

```
void CVICALLBACK EventFunctionName (int menuBarHandle,
                           int menuItemID, void *callbackData,
                           int panelHandle);
```

The event function receives the menubar handle, menu item ID, and panel handle of the menu item that generates the callback. It also receives a pointer to callback data that you define.

If you do not want to process the menu item event through a callback function, supply a value of zero (`0`) for the **eventFunction** parameter.

NewPanel

```
int panelHandle = NewPanel (int parentPanelHandle, char panelTitle[],
                            int panelTop, int panelLeft, int panelHeight,
                            int panelWidth);
```

Purpose

Creates a new top-level panel or a new child panel inside a specific parent panel. It returns a panel handle that you use to specify the new panel in subsequent function calls.

To make the panel a top-level panel, enter `0` for the **parentPanelHandle** parameter.

If you call this function to create a child panel, you must do so in the same thread in which you create the parent panel.

Parameters

Input

Name	Type	Description
parentPanelHandle	integer	Handle of the panel in which to create the new child panel. To make the panel a top-level panel, enter `0`.
panelTitle	string	Title for the new panel.
panelTop	integer	Vertical coordinate at which to place the upper left corner of the panel, directly below the title bar.
panelLeft	integer	Horizontal coordinate at which to place the upper left corner of the panel, directly below the title bar.
panelHeight	integer	Vertical size of the new panel. The height of the panel does not include the title bar or panel frame.
panelWidth	integer	Size of the new panel in window coordinates. The width of the panel does not include the panel frame.

Return Value

Name	Type	Description
panelHandle	integer	Returns a panel handle you use in subsequent function calls to reference the panel. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

Parameter Discussion

The **panelTop** and **panelLeft** coordinates must be integer values from –32,768 to 32,767, 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 32,767.

NewSubMenu

```
int subMenuID = NewSubMenu (int menuBarHandle, int menuItemID);
```

Purpose

Creates a submenu, attaches it to a specific menu item, and returns a submenu ID you use to specify the submenu in subsequent function calls.

A submenu appears as a triangle on the right side of the menu item.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`
menuItemID	integer	Menu item to which to attach the submenu. The menu item ID must be a constant name you assign in the User Interface Editor or a value you obtain from `NewMenuItem`.

Return Value

Name	Type	Description
subMenuID	integer	ID that you use to specify this submenu in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

PlotArc

```
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 onto a graph control.

You define the arc 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`, **begAngle** = `0`, and **arcAngle** = `3600`, then `PlotArc` plots a circle centered on coordinates (100.0, 100.0).

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`
x1	double-precision	Horizontal coordinate of one corner of the rectangle that encloses the arc.
y1	double-precision	Vertical coordinate of one corner of the rectangle that encloses the arc.
x2	double-precision	Horizontal coordinate of the opposing corner of the rectangle that encloses the arc.
y2	double-precision	Vertical coordinate of the opposing corner of the rectangle that encloses the arc.
beginAngle	integer	Starting angle of the arc, in tenths of degrees. Positive angles proceed counter-clockwise and negative angles proceed clockwise, from 0 to 3,600.

Name	Type	Description
arcAngle	integer	Sweep angle of the arc, in tenths of degrees. Positive angles proceed counter-clockwise and negative angles proceed clockwise, from 0 to 3,600.
color	integer	Specifies the color of the arc. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.
fillColor	integer	Specifies the fill color of the figure.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotBitmap

```
int plotHandle = PlotBitmap (int panelHandle, int controlID, double x,
                             double y, double width, double height,
                             char fileName[]);
```

Purpose

Plots a bitmapped image from a file onto a graph control.

You pass the x- and y-coordinates of the point on the graph at which to place the lower left corner of the image.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`
x	double-precision	X-coordinate of the lower left corner of the bitmapped image.
y	double-precision	Y-coordinate of the lower left corner of the bitmapped image.
width	double-precision	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 stretches or shrinks to fit.
height	double-precision	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 stretches or shrinks to fit.
fileName	string	Name of the file that contains the image.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle 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 that contains no directory path, the file is loaded from the project. If the file is not in the project, it is loaded from the directory that contains the project file.

The valid image types appear in the following list:

`.pcx`	Windows and UNIX
`.bmp`, `.dib`, `.rle`, `.ico`	Windows only
`.wmf`	Windows 95/NT only
`.xwd`	UNIX only

You can pass `NULL` or the empty string for **fileName**. This is useful when you want to define the image at a later point in your program by calling `SetCtrlBitmap` on the plot handle this function returns. You cannot see the plot until you specify a valid image.

See Also

`SetPlotAttribute`, `GetPlotAttribute`, `SetCtrlBitmap`.

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

Draws a solid rectangular plot in a graph control. If you want to apply scaling factors and offsets to the data values, refer to `PlotScaledIntensity`.

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 lower left corner of the plot area is at {0, 0}.

The upper right corner of the plot area is at {**numXpts** – 1, **numYpts** – 1}, where,

numXpts = Number of x Points
numYpts = Number of y Points

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
zArray	numeric array	Array that contains the data values to convert to colors.
numberOfXPoints	integer	Number of points to display along the x-axis in each row.
numberOfYPoints	integer	Number of points to display along the y-axis in each column.
zDataType	integer	Specifies the data type of the elements in **zArray**, as well as the data type of the color map values.
colorMapArray	ColorMapEntry	Array of `ColorMapEntry` structures that specifies how to translate the data values in **zArray** into colors. The maximum number of entries is 255.
hiColor	integer	RGB value to which to translate **zArray** values that are higher than the highest data value in **colorMapArray**.
numberOfColors	integer	Number of entries in **colorMapArray**. Must be less than or equal to 255.
interpColors	integer	Indicates how to assign colors to **zArray** data values that do not exactly match the data values in the **colorMapArray**.
interpPixels	integer	Indicates how to color pixels between the pixels assigned to the **zArray** values.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `SetPlotAttribute`, `GetPlotAttribute`, or `DeleteGraphPlot`. Refer to Appendix A for error codes.

Parameter Discussion

zArray must be one of the following data types, which you specify in **zDataType**:

```
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 appear on the graph depend on the location of the data values in **zArray**. **zArray** must be a two-dimensional array of the following form:

```
Array[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 that consist of the following components:

```
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 */
```

colorMapArray defines how to translate data values in **zArray** into color values. If a data value matches exactly to a data value in one of the `ColorMapEntry` structures, then the function converts it to the corresponding color. Otherwise, the following rules 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 rules 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 two.

You do not have to sort the **colorMapArray** entries.

interpPixels specifies how the function colors pixels between the pixels assigned to the **zArray** values. 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 **colorMapArray**.

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 onto a graph control.

You specify the starting and ending points of the line in terms of x- and y-coordinates of the graph.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
x1	double-precision	Horizontal coordinate of the starting point of the line.
y1	double-precision	Vertical coordinate of the starting point of the line.
x2	double-precision	Horizontal coordinate of the ending point of the line.
y2	double-precision	Vertical coordinate of the ending point of the line.
color	integer	Specifies the color of the line. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotOval

```
int plotHandle = PlotOval (int panelHandle, int controlID, double x1,
                           double y1, double x2, double y2, int color,
                           int fillColor);
```

Purpose

Plots a oval onto a graph control.

You define the oval in terms of a rectangle that encloses it. You define the rectangle by specifying two opposing corners, which you express in terms of x- and y-coordinates of the graph.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
x1	double-precision	Horizontal coordinate of one corner of the rectangle that encloses the oval.
y1	double-precision	Vertical coordinate of one corner of the rectangle that encloses the oval.
x2	double-precision	Horizontal coordinate of the opposing corner of the rectangle that encloses the oval.
y2	double-precision	Vertical coordinate of the opposing corner of the rectangle that encloses the oval.

Name	Type	Description
color	integer	Specifies the color of the oval frame. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.
fillColor	integer	Specifies the fill color of the oval.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotPoint

```
int plothandle = PlotPoint (int panelHandle, int controlID,
                            double xCoordinate, double yCoordinate,
                            int pointStyle, int color);
```

Purpose

Plots a point onto a graph control.

You specify the position of the point in terms of x- and y-coordinates of the graph.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
xCoordinate	double-precision	Horizontal position at which to plot the point.
yCoordinate	double-precision	Vertical coordinate at which to plot the point.
pointStyle	integer	Point style to use.
color	integer	Specifies the color of the point. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotPolygon

```
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 onto a graph control.

You define the polygon by a set of connected x-y points. The last point is automatically connected to the first point to close the polygon.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
xArray	void *	Array that contains the values to plot along the x-axis. The data type must be of the type specified by **xDataType.**
yArray	void *	Array that contains the values to plot along the y-axis. The data type must be of the type specified by **yDataType.**
pointsInPolygon	integer	Number of connected points (corners) in the polygon. At least three corners must exist. **pointsInPolygon** determines the number of corners plotted, even if the number of elements in the x and y arrays is greater than the value of **pointsInPolygon.**
xDataType	integer	Specifies the data type of the x array. Refer to Table 3-46 in Chapter 3 for a list of data types.

Name	Type	Description
yDataType	integer	Specifies the data type of the y array. Refer to Table 3-46 in Chapter 3 for a list of data types.
color	integer	Specifies the color of the polygon frame. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.
fillColor	integer	Specifies the fill color of the polygon.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotRectangle

```
int plotHandle = PlotRectangle (int panelHandle, int controlID, double x1,
                                double y1, double x2, double y2, int color,
                                int fillColor);
```

Purpose

Plots a rectangle onto a graph control.

You specify two opposing corners of the rectangle in terms of x- and y-coordinates of the graph.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
x1	double-precision	Horizontal coordinate of one corner of the rectangle.
y1	double-precision	Vertical coordinate of one corner of the rectangle.
x2	double-precision	Horizontal coordinate of the opposing corner of the rectangle.
y2	double-precision	Vertical coordinate of the opposing corner of the rectangle.

Name	Type	Description
color	integer	Specifies the color of the rectangle frame. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.
fillColor	integer	Specifies the fill color of the rectangle.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotScaledIntensity

```
int plotHandle = 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

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:

$$\{(x = 3 \times \mathbf{xGain} + \mathbf{xOffset}), (y = 2 \times \mathbf{yGain} + \mathbf{yOffset})\}$$

The lower left corner of the plot area is located at

$$\{\mathbf{xOffset}, \mathbf{yOffset}\}$$

The upper right corner of the plot area is located at

$$\{(\mathbf{numXpts} - 1) \times \mathbf{xGain} + \mathbf{xOffset}, (\mathbf{numYpts} - 1) \times \mathbf{yGain} + \mathbf{yOffset}\}$$

where

numXpts = Number of x points
numYpts = Number of y points

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
zArray	numeric array	Array that contains the data values to convert to colors.
numberOfXPoints	integer	Number of points to display along the x-axis in each row.
numberOfYPoints	integer	Number of points to display along the y-axis in each column.
zDataType	integer	Specifies the data type of the elements in **zArray**, as well as the data type of the color map values.
yGain	double-precision	Specifies the scaling factor to apply to the vertical coordinates the **zArray** indices represent.
yOffset	double-precision	Specifies the offset to add to the vertical coordinates the **zArray** indices represent.
xGain	double-precision	Specifies the scaling factor to apply to the horizontal coordinates the **zArray** indices represent.
xOffset	double-precision	Specifies the offset to add to the horizontal coordinates the **zArray** indices represent.
colorMapArray	ColorMapEntry	Array of `ColorMapEntry` structures that specifies how to translate the data values in **zArray** into colors. The maximum number of entries is 255.

Name	Type	Description
hiColor	integer	RGB value to which to translate all **zArray** values that are higher than the highest data value in **colorMapArray**.
numberOfColors	integer	Number of entries in **colorMapArray**. Must be less than or equal to 255.
interpColors	integer	Indicates how to assign colors to **zArray** data values that do not exactly match the data values in the **colorMapArray**.
interpPixels	integer	Indicates how to color the pixels between the pixels assigned to the **zArray** values.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `SetPlotAttribute`, `GetPlotAttribute`, or `DeleteGraphPlot`. Refer to Appendix A for error codes.

Parameter Discussion

zArray must be one of the following data types, which you specify in **zDataType**:

```
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 appear on the graph depend on that location of the data values in **zArray**. **zArray** must be a two-dimensional array of the following form:

```
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 the following location on the graph:

$$\{x \times \mathbf{xGain} + \mathbf{xOffset},\ y \times \mathbf{yGain} + \mathbf{yOffset}\}$$

colorMapArray contains up to 255 `ColorMapEntry` structures that consist of the following components:

```
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 */
```

colorMapArray defines how to translate data values in **zArray** into color values. If a data value matches exactly to a data value in one of the `ColorMapEntry` structures, then the function converts it to the corresponding color. Otherwise, the following rules 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 rules 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 two.

You do not have to sort the **colorMapArray** entries.

interpPixels specifies how the function colors pixels between the pixels assigned to the **zArray** values. 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 array.

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

```
int status = PlotStripChart (int panelHandle, int controlID, void *yArray,
                             int numberOfPoints, int startingIndex,
                             int skipCount, int yDataType);
```

Purpose

Adds one or more points to each trace in a strip chart control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
yArray	void *	Array that contains the values to plot along the y-axis. The data type must be of the type you specify in **yDataType.**
numberOfPoints	integer	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 determines the number of points to plot even if the number of elements in **yArray** is greater than the **numberOfPoints.**
startingIndex	integer	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`.

Name	Type	Description
skipCount	integer	Number of **yArray** elements the function skips after plotting each set of points. The default is `0`.
yDataType	integer	Specifies the data type of the **yArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

Interaction between numberOfPoints and Points per Screen

The right edge of the strip chart is considered a grid line. This grid line can contain a data point. To achieve the best results when repeatedly plotting arrays that exactly fill the width of the strip chart, arrange for the data point that is on the left edge of the strip chart before you plot to be on the right edge of the strip chart after you plot. This synchronizes your plots with the grid lines. You can do this by setting the `ATTR_POINTS_PER_SCREEN` attribute to one greater than the value you pass as **numberOfPoints** to `PlotStripChart`. For example, if you intend to add 100 points to the strip chart each time you call `PlotStripChart`, set `ATTR_POINTS_PER_SCREEN` to 101.

If you fail to do this and your strip chart is Continuous mode, the grid lines drift across the strip chart.

If you fail to do this and your strip chart is in Sweep mode or Block mode, gaps can arise from the right side of the strip chart. The following example illustrates this:

You set `ATTR_POINTS_PER_SCREEN` to 100. You call `PlotStripChart` with 100 points. It plots from 0 to 99. You call `PlotStripChart` again with 100 points. It plots from 100 to 199. In this situation, LabWindows/CVI never draws the curve between 99 and 100.

Instead of calling `SetCtrlAttribute` to set `ATTR_POINTS_PER_SCREEN`, you can use the Points per String control in the Edit Strip Chart dialog box of the User Interface Editor.

skipCount

Suppose you have an array that contains four data sets (A,B,C,D) and you want to plot each A-B pair into a strip chart control. Consider the following example of element ordering:

A B C D A B C D A B C D

In the preceding example, entering a **skipCount** of `2` causes the function to skip each C-D pair before it plots the next A-B pair.

See Also

`PlotStripChartPoint`

PlotStripChartPoint

```
int status = PlotStripChartPoint (int panelHandle, int controlID, double y);
```

Purpose

Provides a simple interface for adding one point to a strip chart that contains only one trace. You can perform the same operation using `PlotStripChart`.

Refer to the help text for the `PlotStripChart` function if you want more information about strip chart plots.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
y	double-precision	Data value to plot along the y-axis.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

PlotText

```
int plotHandle = PlotText (int panelHandle, int controlID,
                           double xCoordinate, double yCoordinate,
                           char text[], int font, int textColor,
                           int backgroundColor);
```

Purpose

Plots a text string onto a graph control.

The origin of the text is the lower left corner of the string. You specify the origin of the text in terms of the x- and y-coordinates of the graph.

If the `ATTR_SHIFT_TEXT_PLOTS` graph attribute is non-zero, the text origin is within the graph area, and the text does not entirely fit within the graph, 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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
xCoordinate	double-precision	Horizontal position at which to place the left edge of the text within the graph.
yCoordinate	double-precision	Vertical position at which to place the bottom edge of the text within the graph.
text	string	String to plot.
font	integer	Selects the text font. Refer to Table 3-5 in Chapter 3 for valid fonts.

Name	Type	Description
textColor	integer	Specifies the color of the plotted text. Table 3-3 in Chapter 3 presents a list of the common colors.
backgroundColor	integer	Specifies the background color of the plotted text. Table 3-3 in Chapter 3 presents a list of the common colors.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotWaveform

```
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 onto a graph control.

`PlotWaveform` scales the values in **yArray** according to **yGain** and **yOffset**. It scales the x-axis timebase according to **initialX** and **xIncrement**. It computes each point in the plot as follows:

$$x_i = (i \times \mathbf{xIncrement}) + \mathbf{initialX}$$

$$y_i = (\mathrm{wfm}_i \times \mathbf{yGain}) + \mathbf{yOffset}$$

where *i* is the index of the point in the waveform array.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
yArray	void *	Array that contains the values to plot along the y-axis. The data type must be of the type you specify in **yDataType.**
numberOfPoints	integer	Number of points to plot.
yDataType	integer	Specifies the data type of **yArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.

Name	Type	Description
yGain	double-precision	Specifies the gain to apply to the waveform (**yArray**) data.
yOffset	double-precision	Specifies a constant offset to add to the waveform (**yArray**) data. The default value is `0.0`.
initialX	double-precision	Specifies the initial value for the x-axis.
xIncrement	double-precision	Specifies the increment along the x-axis for each new point.
plotStyle	integer	Curve style to use when plotting the data points. Refer to Table 3-64 in Chapter 3 for a list of plot styles.
pointStyle	integer	Point style to use when plotting the array. The point style determines the type of marker to draw when the plot style is `VAL_CONNECTED_POINTS` or `VAL_SCATTER`. Refer to Table 3-62 in Chapter 3 for a list of point styles.
lineStyle	integer	Selects the line style. Refer to Table 3-63 in Chapter 3 for a list of line styles.
pointFrequency	integer	Specifies the point interval at which to draw marker symbols when the curve style is `VAL_CONNECTED_POINTS` or `VAL_SCATTER.`
color	integer	Specifies the color of the curve. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

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 of a graph control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
xArray	void *	Array that contains the values to plot along the x-axis. The data type must be of the type you specify in **xDataType**.
numberOfPoints	integer	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**.
xDataType	integer	Specifies the data type of the **xArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.
plotStyle	integer	Curve style to use when plotting the data points. Refer to Table 3-64 in Chapter 3 for a list of plot styles.

Name	Type	Description
pointStyle	integer	Point style to use when plotting the array. The point style determines the type of marker to draw when the plot style is `VAL_CONNECTED_POINTS` or `VAL_SCATTER`. Refer to Table 3-62 in Chapter 3 for a list of point styles.
lineStyle	integer	Selects the line style. Refer to Table 3-63 in Chapter 3 for a list of line styles.
pointFrequency	integer	Specifies the point interval at which to draw marker symbols when the curve style is `VAL_CONNECTED_POINTS` or `VAL_SCATTER.`
color	integer	Specifies the color of the curve. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotXY

```
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 on a graph control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
xArray	void *	Array that contains the values to plot along the x-axis. The data type must be of the type you specify in **xDataType**.
yArray	void *	Array that contains the values to plot along the y-axis. The data type must be of the type you specify in **yDataType**.
numberOfPoints	integer	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**.
xDataType	integer	Specifies the data type of the **xArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.
yDataType	integer	Specifies the data type of the **yArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.

Name	Type	Description
plotStyle	integer	Curve style to use when plotting the data points. Refer to Table 3-64 in Chapter 3 for a list of plot styles.
pointStyle	integer	Point style to use when plotting the array. The point style determines the type of marker to draw when the plot style is `VAL_CONNECTED_POINTS` or `VAL_SCATTER`. Refer to Table 3-62 in Chapter 3 for a list of point styles.
lineStyle	integer	Selects the line style. Refer to Table 3-63 in Chapter 3 for a list of line styles.
pointFrequency	integer	Specifies the point interval at which to draw marker symbols when the curve style is `VAL_CONNECTED_POINTS` or `VAL_SCATTER`
color	integer	Specifies the color of the curve. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PlotY

```
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 on a graph control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in he User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
yArray	void *	Array that contains the values to plot along the y-axis. The data type must be of the type specified by **yDataType**.
numberOfPoints	integer	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**.
yDataType	integer	Specifies the data type of the **yArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.
plotStyle	integer	Curve style to use when plotting the data points. Refer to Table 3-64 in Chapter 3 for a list of plot styles.

Name	Type	Description
pointStyle	integer	Point style to use when plotting the array. The point style determines the type of marker to draw when the plot style is `VAL_CONNECTED_POINTS` or `VAL_SCATTER`. Refer to Table 3-62 in Chapter 3 for a list of point styles.
lineStyle	integer	Selects the line style. Refer to Table 3-63 in Chapter 3 for a list of line styles.
pointFrequency	integer	Specifies the point interval at which to draw marker symbols when the curve style is `VAL_CONNECTED_POINTS` or `VAL_SCATTER.`
color	integer	Specifies the color of the curve to plot. 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. Refer to Table 3-3 in Chapter 3 for a list of the common colors.

Return Value

Name	Type	Description
plotHandle	integer	Handle for the plot. You can pass this handle to `DeleteGraphPlot` to delete the individual plot. You can also pass it to `SetPlotAttribute` and `GetPlotAttribute`. Refer to Appendix A for error codes.

PointEqual

```
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 specific points are the same. Returns `0` otherwise.

Parameters

Input

Name	Type	Description
point1	Point	`Point` structure.
point2	Point	`Point` structure.

Return Value

Name	Type	Description
pointsAreEqual	integer	Indication that the two points are the same.

Return Codes

Code	Description
`1`	x- and y-coordinates in the two `Point` structures are the same.
`0`	x- and y-coordinates in the two `Point` structures are not the same.

See Also

`MakePoint`

PointPinnedToRect

```
void PointPinnedToRect (Point point, Rect rect, Point *pinnedPoint);
```

Purpose

Ensures that a point is within the rectangular area you specify. 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 function does not modify the `Point` structure that contains the original location. It stores the calculated location in another `Point` structure.

Parameters

Input

Name	Type	Description
point	Point	`Point` structure that specifies the original location of the point.
rect	Rect	`Rect` structure that specifies the rectangle to which the point is to be pinned.

Output

Name	Type	Description
pinnedPoint	Point	`Point` structure in which to store the calculated location.

Return Value

None

See Also

`MakePoint`, `MakeRect`

PointSet

```
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

Input

Name	Type	Description
xCoordinate	integer	New horizontal location of the point.
yCoordinate	integer	New vertical location of the point.

Output

Name	Type	Description
point	Point	`Point` structure in which to store the new values.

Return Value

None

See Also

`MakePoint`

PostDeferredCall

```
int status = PostDeferredCall (DeferredCallbackPtr deferredFunction,
                               void *callbackData);
```

Purpose

Schedules LabWindows/CVI to call a specific function the next time LabWindows/CVI processes events in the main thread.

You typically use `PostDeferredCall` in a function you install as an asynchronous interrupt handler. In an asynchronous interrupt handler, the types of operations you can perform are limited. For example, you cannot freely access global variables. The **deferredFunction** parameter names the function that contains the code you cannot include in the asynchronous interrupt handler.

This is useful when external devices generate interrupts during source program execution.

Note ***To schedule LabWindows/CVI to call a function in a thread other than the main thread, use*** `PostDeferredCallToThread`**.**

Parameters

Input

Name	Type	Description
deferredFunction	DeferredCallbackPtr	Pointer to the function that LabWindows/CVI invokes the next time it processes events in the main thread.
callbackData	void *	Pointer to data that you define. The pointer is passed to the deferred function.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

The function pointed to by **deferredFunction** takes the following form:

```
void CVICALLBACK DeferredCallbackFunction (void *callbackData);
```

See Also

`PostDeferredCallToThread`

PostDeferredCallToThread

```
int status = PostDeferredCallToThread (DeferredCallbackPtr deferredFunction,
                            void *callbackData,
                            unsigned long targetThreadId);
```

Purpose

Matches the capabilities of `PostDeferredCall`, except that you specify the thread in which LabWindows/CVI invokes the callback function. `PostDeferredCall`, on the other hand, always posts the call to the main thread.

`PostDeferredCallToThread` is useful only in a multithreaded application, and works only on Windows 95/NT. To obtain the ID of a thread under Windows 95 or NT, call the following Windows SDK function when running in that thread:

```
unsigned long GetCurrentThreadId (void);
```

Parameter List

Name	Type	Description
deferredFunction	DeferredCallbackPtr	Pointer to the function that LabWindows/CVI invokes the next time it processes events in the target thread.
callbackData	void *	Pointer to data that you define. The pointer is passed to the deferred function.
targetThreadId	unsigned long	Thread in which LabWindows/CVI calls deferred function.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

The function pointed to by **deferredFunction** takes the following form:

```
void CVICALLBACK DeferredCallbackFunction(void *callbackData);
```

See Also

`PostDeferredCall`

PrintCtrl

```
int printStatus = PrintCtrl (int panelHandle, int controlID, char fileName[],
                             int scaling, int confirmDialogBox);
```

Purpose

Prints the selected control.

While this function is printing, it blocks any other thread in your program that attempts to print.

Remember that LabWindows/CVI maintains only one copy of the print attributes you set with `SetPrintAttribute`. Thus, when you change a print attribute in one thread, the change affects printing functions you subsequently call in other threads.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
fileName	string	Name of the output file. If the name is non-empty, the output is redirected to the file. If the name is not a complete pathname, the file is created relative to the current working directory.
scaling	integer	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 on paper that it has on the screen.
confirmDialogBox	integer	Displays a dialog box before printing to confirm print attributes. Shows the current print attribute values and to what extent the current printer supports them. Allows the end-user to change attribute values during run time.

Return Value

Name	Type	Description
printStatus	integer	Returns the status of the print operation. Refer to Appendix A for error codes.

Return Codes

The `PrintCtrl` function returns a value that contains bit-fields.

Defined Constant	Value
`VAL_TOO_MANY_COPIES`	`(1<<0)`
`VAL_NO_MULTIPLE_COPIES`	`(1<<1)`
`VAL_NO_DUPLEX`	`(1<<2)`
`VAL_NO_LANDSCAPE`	`(1<<3)`
`VAL_CANT_FORCE_MONO`	`(1<<4)`
`VAL_NO_SUCH_XRESOLUTION`	`(1<<5)`
`VAL_NO_MULTIPLE_XRESOLUTIONS`	`(1<<6)`
`VAL_NO_SUCH_YRESOLUTION`	`(1<<7)`
`VAL_NO_MULTIPLE_YRESOLUTIONS`	`(1<<8)`
`VAL_NO_SEPARATE_YRESOLUTION`	`(1<<9)`
`VAL_USER_CANCEL`	`(1<<10)`

See Also

`PrintPanel`, `SetPrintAttribute`, `GetPrintAttribute`

PrintPanel

```
int printStatus = PrintPanel (int panelHandle, char fileName[], int scaling,
                              int scope, int confirmDialogBox);
```

Purpose

Prints the selected panel.

While this function is printing, it blocks any other thread in your program that attempts to print.

Remember that LabWindows/CVI maintains only one copy of the print attributes you set with `SetPrintAttribute`. Thus, when you change a print attribute in one thread, the change affects printing functions you subsequently call in other threads.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
fileName	string	Name of the output file. If the name is non-empty, the output is redirected to the file. If the name is not a complete pathname, the file is created relative to the current working directory.
scaling	integer	Selects the scaling mode for printing. `1` = Expands the object to the size of an entire page. `0` = Prints the object at the same relative location and size on paper that it has on the screen.

Name	Type	Description
scope	integer	Selects the portion of the panel to print: `VAL_VISIBLE_AREA` or `VAL_FULL_PANEL`.
confirmDialogBox	integer	Displays a dialog box before printing to confirm print attributes. Shows the current print attribute values and to what extent the current printer supports them. Allows the end-user to change attribute values during run time.

Return Value

Name	Type	Description
printStatus	integer	Returns the status of the print operation. Refer to Appendix A for error codes.

Return Codes

The `PrintPanel` function returns a value that contains bit-fields.

Defined Constant	Value
`VAL_TOO_MANY_COPIES`	`(1<<0)`
`VAL_NO_MULTIPLE_COPIES`	`(1<<1)`
`VAL_NO_DUPLEX`	`(1<<2)`
`VAL_NO_LANDSCAPE`	`(1<<3)`
`VAL_CANT_FORCE_MONO`	`(1<<4)`
`VAL_NO_SUCH_XRESOLUTION`	`(1<<5)`
`VAL_NO_MULTIPLE_XRESOLUTIONS`	`(1<<6)`
`VAL_NO_SUCH_YRESOLUTION`	`(1<<7)`
`VAL_NO_MULTIPLE_YRESOLUTIONS`	`(1<<8)`
`VAL_NO_SEPARATE_YRESOLUTION`	`(1<<9)`
`VAL_USER_CANCEL`	`(1<<10)`

Parameter Discussion

If **scope** is `VAL_VISIBLE_AREA`, only the portion of the panel that is visible on the screen prints. Menu bars, scroll bars, and a frame are printed along with the visible portion. If **scope** is `VAL_FULL_PANEL`, the entire panel prints. No menubars, scroll bars, or frames print. Regardless of the **scope**, objects within child panels are clipped to the frame of the child panel.

Note ***By default,*** `PrintPanel` ***uses the following settings:***

- `ATTR_PRINT_AREA_WIDTH` ***is set to*** `VAL_USE_ENTIRE_PAPER`
- `ATTR_PRINT_AREA_HEIGHT` ***is set to*** `VAL_INTEGRAL_SCALE`

Note ***If the printout does not fit on the page, call*** `SetPrintAttribute` ***before*** `PrintPanel` ***to change the settings as follows:***

- ***Set*** `ATTR_PRINT_AREA_WIDTH` ***to*** `VAL_INTEGRAL_SCALE`
- ***Set*** `ATTR_PRINT_AREA_HEIGHT` ***to*** `VAL_USE_ENTIRE_PAPER`

See Also

`PrintCtrl`, `SetPrintAttribute`, `GetPrintAttribute`

PrintTextBuffer

```
int status = PrintTextBuffer (char buffer[], char outputFile[]);
```

Purpose

Prints the contents of a text buffer. You can direct the output to the printer or to a file.

Newline/carriage-return characters are honored.

Tabs expand according to the current state of the `ATTR_TAB_INTERVAL` attribute. You can modify this attribute with `SetPrintAttribute`. The default tab interval is four.

Text that extends beyond the end of the paper either truncates or wraps, depending on the state of the `ATTR_TEXT_WRAP` attribute. You can modify this attribute with `SetPrintAttribute`. The default is *wrap*.

While this function is printing, it blocks any other thread in your program that attempts to print.

Remember that LabWindows/CVI maintains only one copy of the print attributes you set with `SetPrintAttribute`. Thus, when you change a print attribute in one thread, the change affects printing functions you subsequently call in other threads.

Parameters

Input

Name	Type	Description
buffer	string	Text to print. Must be terminated by an ASCII NUL byte.
outputFile	string	Name of a file to which to direct the output. If `""`, the output is directed to the printer.

Return Value

Name	Type	Description
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`, `SetPrintAttribute`, `GetPrintAttribute`

PrintTextFile

```
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 expand according to the current state of the `ATTR_TAB_INTERVAL` attribute. You can modify this attribute with `SetPrintAttribute`. The default tab interval is four.

Text that extends beyond the end of the paper either truncates or wraps, depending on the state of the `ATTR_TEXT_WRAP` attribute. You can modify this attribute with `SetPrintAttribute`. The default setting is *wrap*.

While this function is printing, it blocks any other thread in your program that attempts to print.

Remember that LabWindows/CVI maintains only one copy of the print attributes you set with `SetPrintAttribute`. Thus, when you change a print attribute in one thread, the change affects printing functions you subsequently call in other threads.

Parameters

Input

Name	Type	Description
fileName	string	Name of the file to print.
outputFile	string	Name of a file to which to direct the output. If `""`, the output is directed to the printer.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

fileName can be a complete pathname or a simple filename. If the name is a simple filename that contains no directory path, the file is loaded from the project. If the file is not in the project, it is loaded from the directory that contains the project file.

If **outputFile** is a not a complete pathname, it is created relative to the current working directory.

See Also

`PrintTextBuffer`, `SetPrintAttribute`, `GetPrintAttribute`

ProcessDrawEvents

```
int status = ProcessDrawEvents(void);
```

Purpose

When your program executes in a callback function or in code that does not call `RunUserInterface` or `GetUserEvent`, LabWindows/CVI does not update the user interface. If a particular function is overly time-consuming, it essentially “locks out” user interface updates. To allow LabWindows/CVI to process these updates, call `ProcessDrawEvents`.

☞ **Note** ***LabWindows/CVI automatically updates the user interface in*** `GetUserEvent` ***or when a callback returns.***

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

ProcessSystemEvents

```
int status = ProcessSystemEvents (void);
```

Purpose

When you program executes in a callback function or in code that does not call `RunUserInterface` or `GetUserEvent`, LabWindows/CVI does not process user interface and system events. If a particular function is overly time-consuming, it essentially “locks out” user interface and system events. To allow LabWindows/CVI to process events, call `ProcessSystemEvents`. Take care when using `ProcessSystemEvents`, because it can cause other callback functions to execute before it completes.

This function processes all pending events.

☞ **Note** ***When*** `ProcessSystemEvents` ***handles an event that starts a tracking loop, such as the user pulling down a menu,*** `ProcessSystemEvents` ***does not return until the tracking loop completes. In the case of pulling down a menu, the tracking loop does not complete until the user dismisses the menu. Consequently, do not call*** `ProcessSystemEvents` ***in your program if you want to continue executing the subsequent lines of code even during a tracking loop.***

PromptPopup

```
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 enter a reply.

Parameters

Input

Name	Type	Description
title	string	Title to display on the dialog box.
message	string	Message to display on the dialog box. The `\n` character can be used to create multi-line messages.
maxResponseLength	integer	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 NUL byte.

Output

Name	Type	Description
responseBuffer	string	Buffer in which to store the user's response. The buffer must be large enough to hold **maxResponseLength** bytes plus one ASCII NUL byte.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

QueueUserEvent

```
int status = QueueUserEvent (int eventNumber, int panelHandle,
                             int controlID);
```

Purpose

Inserts a programmer-defined event in the `GetUserEvent` queue. You later retrieve the event by calling `GetUserEvent`.

Event numbers 1,000 to 10,000 are reserved for programmer-defined events. You can assign any meaning you choose to the event number.

Parameters

Input

Name	Type	Description
eventNumber	integer	**eventNumber** is an integer value between 1,000 and 10,000 that this function places in the event queue and that you can later retrieve from `GetUserEvent.`
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`. Pass `0` (zero) when the event does not apply to a particular panel.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`. Pass `0` (zero) when the event does not apply to a particular control.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

QuitUserInterface

```
int status = QuitUserInterface (int returnCode);
```

Purpose

Causes `RunUserInterface` to return with a specific return code and thereby terminates event processing. Call `QuitUserInterface` only from within a callback function invoked during execution of `RunUserInterface`.

Parameter

Input

Name	Type	Description
returnCode	integer	Value that the current call to `RunUserInterface` returns when it terminates. Use this value as a flag to pass information back through `RunUserInterface`. Pass a value greater than or equal to zero.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

RecallPanelState

```
int status = RecallPanelState (int panelHandle, char filename[],
                               int stateIndex);
```

Purpose

Reads a panel state from a file you previously created with `SavePanelState`. If the panel is currently visible, all controls are updated to reflect their new states.

Note ***If you have modified the panel in the User Interface Editor or programmatically since you saved the panel state, recalling the panel state can fail or erroneously change the current state of the panel.***

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
filename	string	Name of the file in which you saved the panel state. If the name is a simple filename that contains no directory path, then the file is loaded from the directory that contains the project.
stateIndex	integer	The state index you assigned to the panel state when you saved it with `SavePanelState`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

RectBottom

```
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:

$$\text{bottom} = \text{rect.top} + \text{rect.height}$$

Parameter

Input

Name	Type	Description
rect	Rect	Specifies the location and size of a rectangle.

Return Value

Name	Type	Description
bottom	integer	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

```
void RectCenter (Rect rect, Point *center);
```

Purpose

Calculates the location of the center point of a rectangle. For even heights (or widths), the center point is rounded towards the top (or left).

Parameters

Input

Name	Type	Description
rect	Rect	Specifies the location and size of a rectangle.

Output

Name	Type	Description
center	Point	Specifies the location of the center of the rectangle.

Return Value

None

RectContainsPoint

```
int containsPoint = RectContainsPoint (Rect rect, Point point);
```

Purpose

Returns `1` if the rectangle you specify encloses the point you specify. 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

Input

Name	Type	Description
rect	Rect	Specifies the location and size of a rectangle.

Output

Name	Type	Description
point	Point	Specifies the location of the center of the rectangle.

Return Value

Name	Type	Description
containsPoint	integer	Indicates if **rect** contains **point**.

Return Codes

Code	Description
`1`	**point** is in the interior or on the frame of the rectangle specified by **rect**.
`0`	**point** is outside the frame of the rectangle specified by **rect**.

RectContainsRect

```
int containsRect = RectContainsRect (Rect rect1, Rect rect2);
```

Purpose

Returns `1` if the first rectangle you specify encloses the second rectangle you specify. 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

Input

Name	Type	Description
rect1	Rect	Specifies the location and size of a rectangle.
rect2	Rect	Specifies the location and size of a rectangle.

Return Value

Name	Type	Description
containsRect	integer	Indicates if **rect1** encloses **rect2**.

Return Codes

Code	Description
`1`	**rect1** encloses **rect2**.
`0`	**rect1** does not enclose **rect2**.

RectEmpty

```
int isEmpty = RectEmpty (Rect rect);
```

Purpose

Returns `1` if the rectangle you specify 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

Name	Type	Description
rect	Rect	Specifies the location and size of a rectangle. A `Rect` structure.

Return Value

Name	Type	Description
isEmpty	integer	Indicates if the rectangle **rect** specifies is empty.

Return Codes

Code	Description
`1`	Either rect.height or rect.width is less than or equal to zero.
`0`	Both rect.height and rect.width are greater than zero.

RectEqual

```
int areEqual = RectEqual (Rect rect1, Rect rect2);
```

Purpose

Returns `1` if the location and size of the two rectangles you specify are identical. Returns `0` otherwise.

Parameters

Input

Name	Type	Description
rect1	Rect	Specifies the location and size of a rectangle.
rect2	Rect	Specifies the location and size of a rectangle.

Return Value

Name	Type	Description
areEqual	integer	Indicates if the top, left, height, and width values in **rect1** are identical to those of **rect2**.

Return Codes

Code	Description
`1`	**rect1** and **rect2** have the identical top, left, height, and width values.
`0`	**rect1** and **rect2** do not have the identical top, left, height, and width values.

RectGrow

```
void RectGrow (Rect *rect, int dx, int dy);
```

Purpose

Modifies the values in a `Rect` structure so that the rectangle it defines enlarges or shrinks around its current center point.

Parameters

Input/Output

Name	Type	Description
rect	Rect	On input, specifies the size and location of a rectangle. On output, specifies a rectangle of a different size but the same center point.

Input

Name	Type	Description
dx	integer	Amount to enlarge the rectangle horizontally. Use a negative value to shrink the rectangle horizontally.
dy	integer	Amount to enlarge the rectangle vertically. Use a negative value to shrink the rectangle vertically.

Return Value

None

RectIntersection

```
int rectsIntersect = RectIntersection (Rect rect1, Rect rect2,
                              Rect *intersectionRect);
```

Purpose

Returns an indication of whether the two rectangles you specify intersect. If they do, the function fills in a `Rect` structure describing the intersection area.

Parameters

Input

Name	Type	Description
rect1	Rect	Specifies the location and size of a rectangle.
rect2	Rect	Specifies the location and size of a rectangle.

Output

Name	Type	Description
intersectionRect	Rect	`Rect` structure that specifies the largest rectangle enclosed by both **rect1** and **rect2**. If **rect1** and **rect2** do not intersect, this parameter is set to an empty rectangle with height and width of zero. You can pass `NULL` for this parameter.

Return Value

Name	Type	Description
rectsIntersect	integer	Indicates if **rect1** and **rect2** intersect, in other words, have any points in common.

Return Codes

Code	Description
`1`	**rect1** and **rect2** intersect.
`0`	**rect1** and **rect2** do not intersect.

RectMove

```
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 point you specify.

Parameters

Input

Name	Type	Description
point	Point	`Point` structure specifying the new location of the top, left corner of the rectangle.

Input/Output

Name	Type	Description
rect	Rect	On input, specifies the size and location of a rectangle. On output, specifies a rectangle of the same size but a different location.

Return Value

None

RectOffset

```
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

Input

Name	Type	Description
dx	integer	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.
dy	integer	Amount to shift the rectangle vertically. Use a positive value to shift the rectangle down. Use a negative value to shift the rectangle up.

Input/Output

Name	Type	Description
rect	Rect	On input, specifies the size and location of a rectangle. On output, specifies a rectangle of the same size but a different location.

Return Value

None

RectRight

```
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:

$$\mathbf{rightEdge} = \text{rect.left} + \text{rect.width}$$

Parameters

Input

Name	Type	Description
rect	Rect	Specifies a rectangle.

Return Value

Name	Type	Description
rightEdge	integer	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.

See Also

`RectBottom`

RectSameSize

```
int areSameSize = RectSameSize (Rect rect1, Rect rect2);
```

Purpose

Returns `1` if the two rectangles you specify have the same height and width. Returns `0` otherwise.

Parameters

Input

Name	Type	Description
rect1	Rect	Specifies the location and size of a rectangle.
rect2	Rect	Specifies the location and size of a rectangle.

Return Value

Name	Type	Description
areSameSize	integer	Indicates if the height and width values in **rect1** are identical to those of **rect2**.

Return Codes

Code	Description
`1`	**rect1** and **rect2** have the identical height and width.
`0`	**rect1** and **rect2** do not have the identical height and width.

RectSet

```
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

Input

Name	Type	Description
top	integer	New location of the top edge of the rectangle.
left	integer	New location of the left edge of the rectangle.
height	integer	New height of the rectangle.
width	integer	New width of the rectangle.

Output

Name	Type	Description
rect	Rect	`Rect` structure in which to store the new values.

Return Value

None

See Also

`MakeRect`

RectSetBottom

```
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 location you specify. The bottom edge of the rectangle is **not** enclosed by the rectangle and is equal to the top plus the height.

Parameters

Input

Name	Type	Description
bottom	integer	Y-coordinate of the new bottom edge.

Input/Output

Name	Type	Description
rect	Rect	On input, specifies the size and location of a rectangle. On output, specifies the same rectangle except with a different bottom edge.

Return Value

None

RectSetCenter

```
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 point you specify.

Parameter

Input

Name	Type	Description
center	Point	`Point` structure specifying the location of the new center point of the rectangle.

Input/Output

Name	Type	Description
rect	Rect	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.

Return Value

None

RectSetFromPoints

```
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 the two points you specify. Each point is located on a corner of the frame of the rectangle.

Parameters

Input

Name	Type	Description
point1	Point	Specifies the location of a point.
point1	Point	Specifies the location of a point.

Output

Name	Type	Description
rect	Rect	`Rect` structure that is set to enclose the specified points.

Return Value

None

RectSetRight

```
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 location you specify. The right edge of the rectangle is **not** enclosed by the rectangle and is equal to the left edge plus the width.

Parameter

Input

Name	Type	Description
right	integer	X-coordinate of the new right edge.

Input/Output

Name	Type	Description
rect	Rect	On input, specifies the size and location of a rectangle. On output, specifies the same rectangle except with a different right edge.

Return Value

None

RectUnion

```
void RectUnion (Rect rect1, Rect rect2, Rect *unionRect);
```

Purpose

Calculates the smallest rectangle that encloses the two rectangles you specify.

Parameters

Input

Name	Type	Description
rect1	Rect	Specifies the size and location of a rectangle.
rect2	Rect	Specifies the size and location of a rectangle.

Output

Name	Type	Description
unionRect	Rect	`Rect` structure that specifies the smallest rectangle that encloses both **rect1** and **rect2**.

Return Value

None

RefreshGraph

```
int status = RefreshGraph (int panelHandle, int controlID);
```

Purpose

Immediately redraws the plot area.

This action removes from the screen any plots that you delete using `DeleteGraphPlot` in delayed draw mode. It also displays any plots that you plot while `ATTR_REFRESH_GRAPH` is FALSE.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

RegisterWinMsgCallback

```
int messageNumber = RegisterWinMsgCallback
                           (WinMsgCallbackPtr callbackFunction,
                           char messageIdentifier[], void *callbackData,
                           int dataSize, int *callbackID,
                           int deleteWhenProgramStops);
```

☞ **Note** ***Only the Windows versions of LabWindows/CVI support*** `RegisterWinMsgCallback`**.**

Purpose

Registers a callback function that LabWindows/CVI calls when your application receives a specific Windows message. `RegisterWinMsgCallback` returns the actual number of the Windows message.

You can call the Windows SDK `PostMessage` function from a DLL or another application to send the Windows message to LabWindows/CVI. The window handle that you pass to `PostMessage` determines the thread in which the callback function is invoked. To send a message to a particular thread, pass the window handle that `GetCVIWindowHandle` returns when you call it from that thread.

Parameters

Input

Name	Type	Description
callbackFunction	WinMsgCallbackPtr	Name of the user function that LabWindows/CVI calls whenever it receives the Windows message number that `RegisterWinMsgCallback` returns. To send a message to LabWindows/CVI, call the Windows API function `PostMessage` from a DLL or another application.
messageIdentifier	string	A string you specify that allows two processes to use the same Windows message number. Refer to the following discussion.
callbackData	void *	Pointer to data that you define. This pointer or a pointer to a copy of the data is passed to the event function.

Name	Type	Description
dataSize	integer	If **dataSize** = zero, the **callbackData** pointer is the same pointer that **callbackFunction** receives. If **dataSize** is greater than zero, this function copies the data pointed to by the **callbackData** pointer, and the **callbackFunction** receives a pointer to the copy as its **callbackData** parameter.
deleteWhen Program Stops	integer	`0` = The callback function remains installed when your program terminates. Non-Zero = LabWindows/CVI automatically calls `UnRegisterWinMsgCallback` when your program terminates.

Output

Name	Type	Description
callbackID	integer	ID that you pass to `UnRegisterWinMsgCallback` to disable the callback function.

Return Value

Name	Type	Description
messageNumber	integer	Message number that Windows assigns. Zero indicates that the function failed.

Parameter Discussion

callbackFunction is a pointer to an event function that takes the following form:

```
void CVICALLBACK EventFunctionName (WinMsgWParam wParam,
             WinMsgLParam lParam, void *callbackData);
```

The event function receives the **wParam** and **lParam** parameters that were passed to `PostMessage`. The **lParam** parameter is always an unsigned long integer. The **wParam** parameter is an unsigned short integer under Windows 3.1 and an unsigned 32-bit integer under Windows 95/NT. **callbackData** is the same **callbackData** pointer that you pass to `RegisterWinMsgCallback` or a pointer to a copy of the data the original pointer pointed to.

If you pass zero (`0`) for the **messageIdentifier**, `RegisterWinMsgCallback` returns a unique Windows message number. Subsequent calls to `RegisterWinMsgCallback`, or the Windows API function `RegisterWindowMessage`, do not return the same **messageNumber** until you call `UnRegisterWinMsgCallback` with the **callbackId** that `RegisterWinMsgCallback` also returns.

If you pass a string for the **messageIdentifier**, any subsequent call to `RegisterWinMsgCallback`, or the Windows API function `RegisterWindowMessage`, that uses the same **messageIdentifier** string returns the same **messageNumber**.

To send the message you registered through `RegisterWinMsgCallback` to LabWindows/CVI, pass the **messageNumber** this function returns to the Windows API function `PostMessage`.

If `RegisterWinMsgCallback` fails, **messageNumber** is `0`.

See Also

`UnRegisterWinMsgCallback, GetCVIWindowHandle`

RemovePopup

```
int status = RemovePopup (int removePopup);
```

Purpose

Removes either the active pop-up panel or all pop-up panels.

You must call this function from the same thread in which you call `InstallPopup`.

Parameter

Input

Name	Type	Description
removePopup	integer	Selects whether to remove all pop-up panels or only the active pop-up panel. `1` = All `0` = Active only

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`InstallPopup`

ReplaceAxisItem

```
int status = ReplaceAxisItem (int panelHandle, int controlID,
                              int axis, int itemIndex, char itemLabel[],
                              double itemValue);
```

Purpose

Replaces the string/value pair at a specific 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, set the `ATTR_XUSE_LABEL_STRINGS` attribute to TRUE. To see string labels on a y-axis, set the `ATTR_YUSE_LABEL_STRINGS` attribute to TRUE.

You create the original list of label strings in the User Interface Editor or by calling `InsertAxisItem`.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID returned by `NewCtrl` or `DuplicateCtrl`.

axis	integer	Specifies the axis for which to replace the string/value pair at the index you specify. Valid values:
		`VAL_XAXIS`
		`VAL_LEFT_YAXIS`
		`VAL_RIGHT_YAXIS` (graphs only)

itemIndex	integer	Zero-based index of the item to replace.

Name	Type	Description
itemLabel	string	String to replace the existing string in the item at the index you specify. If you pass `0`, the existing string is not replaced. An axis label displays a maximum of 31 characters.
itemValue	double-precision	Value to replace the existing value in the string/value pair at the index you specify. The string appears as an axis label at the location of the value.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`InsertAxisItem`, `DeleteAxisItem`, `ClearAxisItems`, `GetNumAxisItems`

ReplaceListItem

```
int status = ReplaceListItem (int panelHandle, int controlID, int itemIndex,
                              char itemLabel[], ...);
```

Purpose

This function replaces the label/value pair at a specific index in a list control with a new label/value pair.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list of the item to replace.
itemLabel	string	Label to associate with the new value. Pass `0` to use the existing label.
itemValue	depends on the data type of the list control	Value to associate with the new label. The data type must be the same as the data type of the control.

Return Value

Name	Type	Description
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 can be a complete pathname or a simple filename. If it is a simple filename, the image file must be in the project or in the directory of the project. If you pass `NULL` or the empty string, LabWindows/CVI creates a placeholder for the image that you can fill using `ReplaceListitem` or `SetImageBits`.

See Also

`InsertListItem, DeleteListItem, ClearListCtrl`

ReplaceTextBoxLine

```
int status = ReplaceTextBoxLine (int panelHandle, int controlID,
                                 int lineIndex, char text[]);
```

Purpose

Replaces a specific text box line with the string you specify.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
lineIndex	integer	Zero-based index into the text box.
text	string	String to replace the line of text at the **lineIndex**.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

ResetTextBox

```
int status = ResetTextBox (int panelHandle, int controlID, char text[]);
```

Purpose

Replaces all text in a text box with the string you specify.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
text	integer	String to replace all text in the text box.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

ResetTimer

```
int status = ResetTimer (int panelHandle, int controlID);
```

Purpose

Resets the interval start times for timer controls. `ResetTimer` resets a timer whether or not it is disabled or suspended. When you reset a timer with an `ATTR_INTERVAL` of *x* seconds, `ResetTimer` reschedules the interval to end *x* seconds from the time of the call.

You can specify an individual timer control on a panel, all timer controls on a panel, or all timer controls on all panels in the current thread.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`. Pass `0` to indicate all timer controls on all panels in the current thread.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`. Pass `0` to indicate all timer controls on the specified panel.

Return Value

Name	Type	Description
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`.

This function affects only timer controls you loaded or created in the active thread.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`SuspendTimerCallbacks`, `ResetTimer`

RunPopupMenu

```
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.

If the user selects an item from the menu, the following actions occur:

- If a callback function is associated with the menu item, `RunPopupMenu` calls the function.
- `RunPopupMenu` returns the ID of the menu item the user selected.

In most cases, call this function from a User Interface panel or control callback function when it receives a `LEFT_CLICK`, `RIGHT_CLICK`, or `KEYPRESS` event.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.
menuID	integer	ID for a particular menu within a menubar. The Menu ID should be a constant name, located in the `.uir` header file, generated in the User Interface Editor, or a value you obtain from `NewMenu`.
panelHandle	integer	Handle for the panel over which you want the menu to appear. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
top	integer	Vertical coordinate at which to place the upper left corner of the menu. Must be a value from –32,768 to 32,767. It represents the pixel offset from the top of the panel that **panelHandle** specifies.

Name	Type	Description
left	integer	Horizontal coordinate at which to place the upper left corner of the menu. Must be a value from –32,768 to 32,767. It represents the pixel offset from the left edge of the panel that **panelHandle** specifies.
pinTop	integer	Vertical coordinate of the upper left corner of the “pin” area relative to the upper left corner of the panel that **panelHandle** specifies.
pinLeft	integer	Horizontal coordinate of the upper left corner of the “pin” area relative to the upper left corner of the panel that **panelHandle** specifies.
pinHeight	integer	Height of the “pin” area.
pinWidth	integer	Width of the “pin” area.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Return Codes

Code	Description
`>0`	ID of menu item selected.
`0`	User did not select a menu item.

Parameter Discussion

pinTop, **pinLeft**, **pinHeight**, and **pinWidth** define the *pin* area. Usually the pin area is the area on which the user clicked on the mouse button in order to access 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 a pin area, pass zeros for **pinTop**, **pinLeft**, **pinHeight**, and **pinWidth**.

RunUserInterface

```
int status = RunUserInterface (void);
```

Purpose

`RunUserInterface` runs the User Interface and issues events to callback functions.

`RunUserInterface` does not return until you call `QuitUserInterface` from within a callback function. `RunUserInterface` returns the value that you pass to `QuitUserInterface`.

Return Value

Name	Type	Description
status	integer	Value that you pass to `QuitUserInterface`.

SavePanelState

```
int status = SavePanelState (int panelHandle, char filename[],
                             int stateIndex);
```

Purpose

Saves the current values of all controls on a panel.

Also saves the following control attributes:

- Label/value pairs and their indices
- Minimum, maximum, and increment values for numerics
- List box checkmark state values

If you want to retain the data for plots in a graph control, your original array must still be in memory when you call `RecallPanelState`. Alternatively, you can configure the graph to make a copy of your plot data. You can do this in the User Interface Editor or by calling `SetGraphAttribute` with the `ATTR_COPY_ORIGINAL_DATA` attribute.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
filename	integer	Name of the file in which to save the panel state. If the name is a simple filename that contains no directory path, the file is saved in the directory containing the project.
stateIndex	integer	Assigns a unique state index to each panel state so that you can save multiple panel states to the same file. If the file already contains a panel with the same state index, `SavePanelState` overwrites it.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetActiveCtrl

```
int status = SetActiveCtrl (int panelHandle, int controlID);
```

Purpose

Sets the active control on a panel.

The active control is the control that receives keyboard events when the panel is the active panel.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetActiveGraphCursor

```
int status = SetActiveGraphCursor (int panelHandle, int controlID,
                                   int activeCursorNumber);
```

Purpose

Sets the active cursor on a graph control.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
activeCursorNumber	integer	Specifies the new active cursor number. The value can range from 1 to the number of cursors for the graph. You set the number of cursors in the User Interface Editor or through `SetCtrlAttribute.`

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetActivePanel

```
int status = SetActivePanel (int panelHandle);
```

Purpose

Makes the panel you specify the active panel. The active panel is the panel that receives keyboard events.

Parameter

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetAxisRange

```
int status = SetAxisRange (int panelHandle, int controlID, int xAxisScaling,
                           double xminORxinit, double xmaxORxinc,
                           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.

`SetAxisRange` does not work on the right y-axis and is therefore obsolete. It is recommended that you use `SetAxisScalingMode` instead. To set the x-offset and x-increment for a strip chart, use the `ATTR_XAXIS_GAIN` and `ATTR_XAXIS_OFFSET` attributes.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
xAxisScaling	integer	Scaling mode for the x-axis. Table 4-7 lists valid values.
xminORxinit	double-precision	For a graph, **xmin** specifies the minimum axis range when you configure the x-axis for manual scaling. In this case, **xmax** must exceed **xmin**. For a strip chart, **xinit** specifies the initial x-axis value.
xmaxORxinc	double-precision	For a graph, **xmax** specifies the maximum axis range when you configure the x-axis for manual scaling. In this case, **xmax** must exceed **xmin**. For a strip chart, **xinc** specifies the x-axis increment for each new point.

Name	Type	Description
yAxisScaling	integer	Scaling mode for the y-axis. Table 4-8 lists valid values.
ymin	double-precision	Specifies the minimum axis range when you configure the y-axis for manual scaling. In this case, **ymax** must exceed **ymin**.
ymax	double-precision	Specifies the maximum axis range when you configure the y-axis for manual scaling. In this case, **ymax** must exceed **ymin**.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

xAxisScaling

Table 4-7. xAxisScaling Valid Values

Valid Values	Description
`VAL_NO_CHANGE`	Current x-axis scaling mode remains unchanged. **xmin** and **xmax** are not used.
`VAL_MANUAL`	X-axis is manually scaled, and **xmin** and **xmax** define its range.
`VAL_AUTOSCALE`	X-axis is autoscaled. **xmin** and **xmax** are not used. You cannot use `VAL_AUTOSCALE` in strip charts.
`VAL_LOCK`	X-axis is manually scaled using the current axis range. You cannot use `VAL_LOCK` in strip charts.

yAxisScaling

Table 4-8. yAxisScaling Valid Values

Valid Values	Description
`VAL_NO_CHANGE`	Current y-axis scaling mode remains unchanged. **ymin** and **ymax** are not used.
`VAL_MANUAL`	Y-axis is manually scaled, and **ymin** and **ymax** define its range.
`VAL_AUTOSCALE`	Y-axis is autoscaled. **ymin** and **ymax** are not used. You cannot use `VAL_AUTOSCALE` in strip charts.
`VAL_LOCK`	Y-axis is manually scaled using the current axis range. You cannot use `VAL_LOCK` in strip charts.

SetAxisScalingMode

```
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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
axis	integer	Specifies for which axis to set the mode and range. Valid values: `VAL_XAXIS` (graphs only) `VAL_LEFT_YAXIS` (graphs and strip charts) `VAL_RIGHT_YAXIS` (graphs only)
axisScaling	integer	Scaling mode for the axis. Table 4-9 lists valid values.
min	double-precision	Minimum axis value when you configure the axis for manual scaling.
max	double-precision	Maximum axis value when you configure the axis for manual scaling.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

axisScaling must be one of the values shown in Table 4-9.

Table 4-9. axisScaling Valid Values

Valid Value	Description
`VAL_MANUAL`	Axis is manually scaled, and **min** and **max** define its range.
`VAL_AUTOSCALE`	Axis is autoscaled. **min** and **max** are not used. You cannot use `VAL_AUTOSCALE` in strip charts.
`VAL_LOCK`	Axis is manually scaled using the current, usually autoscaled, minimum and maximum values on the axis. You cannot use `VAL_LOCK` in strip charts.

If **axisScaling** is `VAL_MANUAL`**, max** must exceed **min**.

See Also

`GetAxisScalingMode`

SetBitmapData

```
int status = SetBitmapData (int bitmapID, int bytesPerRow, int pixelDepth,
                            int colorTable[], unsigned char bits[],
                            unsigned char mask[]);
```

Purpose

Changes the image contents of an existing bitmap. The width and height of the bitmap do not change. By using `SetBitmapData`, you can achieve better performance than when you call `DiscardBitmap` and `NewBitmap`.

If the bitmap already has a mask, you can either supply the same **mask** array, supply a new **mask** array, or pass `REMOVE_TRANSPARENCY_MASK` to remove the mask. When you use `REMOVE_TRANSPARENCY_MASK`, the color of the pixels that were transparent under the old mask are unpredictable. You can set the value of these pixels in the **bits** parameter.

If the bitmap does not already have a mask, you can pass `NULL`, or you can add a mask by supplying a **mask** array.

You can use `SetBitmapData` with bitmaps created through `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`. You cannot use `SetBitmapData` if the bitmap originated from a Windows metafile (`.wmf`).

Parameter List

Name	Type	Description
bitmapID	integer	ID of the bitmap that contains the image. You must obtain the ID from one of the following functions: `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`.
bytesPerRow	integer	Number of bytes on each scan line of the image.
pixelDepth	integer	Number of bits per pixel.
colorTable	integer array	Array of RGB color values, or `NULL` if **pixelDepth** is greater than eight.
bits	unsigned char array	Array of bits that determine the colors to be displayed on each pixel in the image.
mask	unsigned char array	Array that contains one bit per pixel in the image. Each bit specifies whether to actually draw the pixel. Can be `NULL` or `REMOVE_TRANSPARENCY_MASK`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

Depending on the **pixelDepth** and on the width of the bitmap, the number of bits per line in the **bits** array might not be an even multiple of eight. If not, then the extra bits required to reach the next byte boundary are called *padding*. If you specify **bytesPerRow** as a positive number, then the bits for each scan line must start on a byte boundary, and so you might have to use padding. In fact, you can set **bytesPerRow** to be larger than the minimum number of bytes you actually require. You can consider these extra bytes to be padding, too. If you pass `-1`, no padding occurs. 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 eight or less, the number of entries in the **colorTable** array must equal two raised to the power of the **pixelDepth** parameter. The **bits** array contains indices into the **colorTable** array. If the pixelDepth is greater than eight, 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 in the **bits** array is a 3-byte RGB value of the form `0xRRGGBB`, where `RR`, `GG`, and `BB` represent the red, green, and blue value of the color. The `RR` byte is always at the lowest memory address of the three bytes.

For a **pixelDepth** of `32`, each pixel in the **bits** array is a 4-byte RGB value of the form `0x00RRGGBB`, where `RR`, `GG`, and `BB` represent the red, green, and blue value 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 is always in the least significant byte. On little-endian platforms, for example, Intel processors, `BB` is at the lowest memory address. On big-endian platforms, for example, Motorola processors, `BB` is at the highest address. Notice that this byte ordering scheme differs from the byte ordering scheme 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. An exception applies when the **pixelDepth** is `1`. In this case, the pixels that have a **bits** value of `1`, called foreground pixels, are always drawn, and the **mask** affects only the pixels that have a value of `0`, called background pixels.

You must pad each row of the **mask** to the nearest even-byte boundary. For example, if the width of the image is 21 pixels, then each row of the mask must have 32 bits, in other words, four bytes, of data.

See Also

`NewBitmap`, `GetCtrlBitmap`, `AllocBitmapData`, `DiscardBitmap`

SetCtrlAttribute

```
int status = SetCtrlAttribute (int panelHandle, int controlID,
                               int controlAttribute, ...);
```

Purpose

Sets the value of a control attribute.

Control attributes have differing data types and differing valid ranges. A list of attributes, their data types and valid values are provided in Tables 3-9 to 3-44 in Chapter 3, *Programming with the User Interface Library*.

Although you can obtain the value of all control attributes using `GetCtrlAttribute`, there are some control attributes that you cannot modify. The attribute lists in Tables 3-9 to 3-44 of Chapter 3, *Programming with the User Interface Library*, indicate which attributes you cannot modify.

Note ***When you set control attributes that affect the font of a control, modify the*** `ATTR_TEXT_FONT` ***attribute first.***

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
controlAttribute	integer	A particular control attribute.
attributeValue	depends on the attribute	New value of the control attribute.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetCtrlBitmap

```
int status = SetCtrlBitmap (int panelHandle, int controlID, int imageID,
                            int bitmapID);
```

Purpose

Sets an image in a control from a bitmap object. You can use this function to replace an existing image in a control or to create a new image in a control.

The following control types can contain images: picture controls, picture rings, picture buttons, and graph controls.

For picture controls, you can use this function as an alternative to `DisplayImageFile`.

For picture buttons, you can use this function as an alternative to calling `SetCtrlAttribute` on the `ATTR_IMAGE_FILE` attribute.

For picture rings, you can use this function 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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
imageID	integer	For picture rings, the zero-based index of an image in the ring. For graphs, this argument is the **plotHandle** you obtain from `PlotBitmap`. For picture controls and picture buttons, this argument is ignored.
bitmapID	integer	ID of the bitmap object containing the new image. You obtain the ID from `NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `ClipboardGetBitmap`, `GetCtrlDisplayBitmap`, or `GetPanelDisplayBitmap`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`NewBitmap`, `GetBitmapFromFile`, `GetCtrlBitmap`, `GetCtrlDisplayBitmap`, `GetPanelDisplayBitmap`, `ClipboardGetBitmap`

SetCtrlIndex

```
int status = SetCtrlIndex (int panelHandle, int controlID, int itemIndex);
```

Purpose

Sets the current index of a list control. This moves the selection in the list control to the item located at the index you specify.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list control.

Return Value

Name	Type	Description
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 a value you specify.

When you call `SetCtrlVal` on a list box or a ring control, it sets the current list item to the first item that has the value you specify. To set the current list item through a zero-based index, use `SetCtrlIndex`.

When you call `SetCtrlVal` on a text box, it appends **value** to the contents of the text box. Use `ResetTextBox` to replace the contents of the text box with **value**.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
value	Depends on the data type of the control.	New value of the control. The data type of **value** must match the data type of the control.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetCursorAttribute

```
int status = SetCursorAttribute (int panelHandle, int controlID,
                           int cursorNumber, int cursorAttribute,
                           int attributeValue);
```

Purpose

Sets the value of a graph cursor attribute.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
cursorNumber	integer	Identifies the cursor. Can be `1` to the number of cursors for the graph. You set the number of cursors in the User Interface Editor or through `SetCtrlAttribute.`

cursorAttribute	integer	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`

attributeValue	integer	New value of the cursor attribute. Refer to Table 3-58 for a complete listing of cursor attribute values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetFontPopupDefaults

```
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 end-user clicks on 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 you set with this function apply only to the next call to `FontSelectPopup`. If you do not call this function before you call `FontSelectPopup`, the default values have the initial settings values shown in Table 4-10.

Table 4-10. Default Values of SetFontPopupDefaults

Font Pop-Up Defaults	Initial Settings
typefaceName	`VAL_DIALOG_FONT`
bold	`0`
underline	`0`
strikeOut	`0`
italic	`0`
justification	`VAL_LEFT_JUSTIFIED`
textColor	`VAL_BLACK`
fontSize	`12`

LabWindows/CVI maintains a separate set of `FontSelectPopup` default values for each thread of your program.

Parameters

Input

Name	Type	Description
typefaceName	string	Default typeface name, for example, "Courier," for use by `FontSelectPopup`.
bold	integer	Default bold setting for use by `FontSelectPopup`.
underline	integer	Default underline setting for use by `FontSelectPopup`.
strikeOut	integer	Default strike out setting for use by `FontSelectPopup`.
italic	integer	Default italic setting for use by `FontSelectPopup`.
justification	integer	Default justification setting for use by `FontSelectPopup`.
textColor	integer	Default text color setting for use by `FontSelectPopup`.
fontSize	integer	Default font size for use by `FontSelectPopup`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`FontSelectPopup`

SetGraphCursor

```
int status = SetGraphCursor (int panelHandle, int controlID,
                             int cursorNumber, double x, double y);
```

Purpose

Sets the position of a specific graph cursor.

The position is relative to the current range of the x- and y-axes.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
cursorNumber	integer	Identifies the graph cursor. Can be `1` to the number of cursors for the graph. You set the number of cursors in the User Interface Editor or through `SetCtrlAttribute.`
x	double-precision	Specifies the x-coordinate of the new cursor position.
y	double-precision	Specifies the y-coordinate of the new cursor position.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetGraphCursorIndex

```
int status = SetGraphCursorIndex (int panelHandle, int controlID,
                                  int cursorNumber, int plotHandle,
                                  int arrayIndex);
```

Purpose

Attaches a cursor to particular data point that you specify by a plot handle and array index.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
cursorNumber	integer	Identifies the graph cursor. Can be `1` to the number of cursors for the graph. You set the number of cursors in the User Interface Editor or through `SetCtrlAttribute`.

plotHandle	integer	Specifies the handle of the plot on which to attach the cursor. Cursors cannot be attached to plots generated by the following functions:
		`PlotText` `PlotOval`
		`PlotRectangle` `PlotArc`
		`PlotLine` `PlotBitmap`
		`PlotIntensity`

arrayIndex	integer	Specifies the array index of the data point on which to attach the cursor.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetIdleEventRate

```
int status = SetIdleEventRate (int interval);
```

Purpose

Sets the interval between idle events that the main callback receives.

You process idle events in the main callback function you install through `InstallMainCallback`.

Parameter

Input

Name	Type	Description
interval	integer	Specifies the wait interval between idle events, in milliseconds. A value of zero causes 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, plus any operating system latency. The resolution of the system timer is generally 1 ms.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetImageBits

```
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. You can use `SetImageBits` to replace an existing image in a control or to create a new image in a control.

The following control types can contain images: picture controls, picture rings, picture buttons, graph controls.

For a picture control, you can use `SetImageBits` as an alternative to `DisplayImageFile`.

For a picture buttons, you can use `SetImageBits` as an alternative to calling `SetCtrlAttribute` on the `ATTR_IMAGE_FILE` attribute.

For picture rings, you can use `SetImageBits` 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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
imageID	integer	For a picture ring, the zero-based index of an image in the ring. For a graph, the plot handle you obtain from `PlotBitmap`. For picture controls and buttons, this is ignored.
rowBytes	integer	Number of bytes on each scan line of the image.
depth	integer	Number of bits per pixel.
width	integer	Width of the image, in pixels.
height	integer	Height of the image, in pixels.
colorTable	integer array	Array of RGB color values.
bitmap	unsigned char array	Array of bits that determine the colors to be displayed on each pixel in the image.
mask	unsigned char array	Array containing one bit per pixel in the image. Each bit specifies whether to actually draw the pixel.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

If either the **width** or **height** parameter is `-1` an image currently exists, `SetImageBits` deletes the image.

Depending on the depth and width, the number of bits per scan line in the bitmap array might not be an even multiple of eight. 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 you might have to use padding. In fact, you can specify a value for **rowBytes** that is larger than the minimum number of bytes you actually require. The extra bytes are also considered padding. If you pass `-1`, no padding occurs. 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 eight or less, the number of entries in the **colorTable** array must equal two raised to the power of the **pixelDepth** parameter. The **bits** array contains indices into the **colorTable** array. If the **pixelDepth** is greater than eight, 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 in the **bits** array is 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 is always in the least significant byte. On little-endian platforms, for example, Intel processors, `BB` is at the lowest memory address. On big-endian platforms, for example, Motorola processors, `BB` is at the highest address. Notice that this byte ordering scheme differs from the byte ordering scheme 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. An exception applies when the **pixelDepth** is `1`. In this case, the pixels that have a **bits** value of `1`, called foreground pixels, are always drawn, and the **mask** affects only the pixels that have a value of `0`, called background pixels.

You must pad each row of the **mask** to the nearest even-byte boundary. For example, if the width of the image is 21 pixels, then each row of the mask must have 32 bits, in other words, four bytes, of data.

A mask is useful for achieving transparency.

You can pass `NULL` if you do not want a mask.

See Also

`GetImageInfo`, `AllocImageBits`, `GetImageBits`

SetInputMode

```
int status = SetInputMode (int panelorMenuBarHandle,
                           int controlorMenuItemID, int inputMode);
```

Purpose

Controls whether LabWindows/CVI recognizes user input on a panel, control, menubar, menu, or menu item.

When you disable the input mode of an object, it appears dim on the screen.

Parameters

Input

Name	Type	Description
panelorMenuBarHandle	integer	Handle of the panel or menubar on which you want to set the recognition of user input. Use `-1` to specify all panels and all menubars.
controlorMenuItemID	integer	ID of the control or menu item on which you want to set the recognition of user input. The ID is the defined constant you assign to the control or menu item in the User Interface Editor or the ID that you obtain from `NewCtrl` or `NewMenuItem`. Use `-1` to specify all controls on the panel or all items in the menubar.
inputMode	integer	Specifies whether to enable or disable user input. `0` = disable `1` = enable

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetListItemImage

```
int status = SetListItemImage (int panelHandle, int controlID, int itemIndex,
                               int image);
```

Purpose

Places a predefined image in a list control at the left-hand side of the line that contains the list item you specify with a zero-based index.

You cannot use this function on picture ring controls. For picture rings, refer to `SetImageBits`.

Table 4-11. Valid Values for Images

Image	Defined Constant
no folder	`VAL_NO_IMAGE`
folder	`VAL_FOLDER`
open folder	`VAL_OPEN_FOLDER`
current folder	`VAL_CURRENT_FOLDER`

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
itemIndex	integer	Zero-based index into the list.
image	integer	Specifies which image to display. Table 4-11 lists valid values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetMenuBarAttribute

```
int status = SetMenuBarAttribute (int menuBarHandle, int menuOrMenuItemID,
                                  int menuBarAttribute, ...);
```

Purpose

Sets the value of a specific menubar attribute.

Parameters

Input

Name	Type	Description
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.
menuOrMenuItemID	integer	Menu or menu item ID you assign in the User Interface Editor or you obtain from the `NewMenu` or `NewMenuItem`. If the attribute corresponds to the entire menubar, pass `0` for this parameter.
menuBarAttribute	integer	Refer to Table 3-6 in Chapter 3 for a complete listing of menubar attributes.
attributeValue	depends on the attribute	New value of the menubar attribute. Refer to Table 3-6 in Chapter 3 for a complete listing of menubar attributes, their data types, and their valid values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetMouseCursor

```
int status = SetMouseCursor (int mouseCursorStyle);
```

Purpose

Sets the appearance of the mouse cursor to the style you specify.

`SetMouseCursor` sets the mouse cursor appearance for all existing panels, regardless of the thread in which you create them, and any panels you subsequently create by calling `NewPanel`, `DuplicatePanel`, or `LoadPanel`.

Parameter

Input

Name	Type	Description
mouseCursorStyle	integer	Specifies the mouse cursor style. Refer to Table 3-4 in Chapter 3 for a list of valid mouse cursor styles.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetPanelAttribute

```
int status = SetPanelAttribute (int panelHandle, int panelAttribute, ...);
```

Purpose

Sets the value of a particular panel attribute.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
panelAttribute	integer	A particular panel attribute. Refer to Table 3-2 in Chapter 3 for a complete listing of panel attributes.
attributeValue	depends on the attribute	New value of the panel attribute. Refer to Table 3-2 in Chapter 3 for a complete listing of panel attributes, their data types, and their valid values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetPanelMenuBar

```
int status = SetPanelMenuBar (int panelHandle, int menuBarHandle);
```

Purpose

Assigns a menubar to the panel you specify.

A panel can have only one menubar at a time, but multiple panels can share the same menubar.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
menuBarHandle	integer	Specifier for a particular menubar that is currently in memory. You obtain this handle from `LoadMenuBar` or `NewMenuBar`.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetPanelPos

```
int status = SetPanelPos (int panelHandle, int panelTop, int panelLeft);
```

Purpose

Sets the position of the upper left corner of a panel, directly below the title bar, relative to the top left corner of the screen or parent panel.

The size of the panel remains constant. Call `SetPanelSize` to change the size of the panel.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
panelTop	integer	Vertical screen coordinate at which to place the upper left corner of the panel, directly below the title bar.
panelLeft	integer	Horizontal screen coordinate at which to place the upper left corner of the panel, directly below the title bar.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

The **panelTop** and **panelLeft** coordinates must be integer values from –32,768 to 32,767, 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.

SetPanelSize

```
int status = SetPanelSize (int panelHandle, int height, int width);
```

Purpose

Sets the height and width of the panel. The top and left edges of the panel remain constant.

Although you can change the height or width independently using `SetPanelAttribute`, you must use `SetPanelSize` if you want to change both the height and width of a panel for which you have enabled the `ATTR_SCALE_CONTENTS_ON_RESIZE` attribute. When you change the height and width in one call, `SetPanelSize` scales and redraws the panel only once.

To change the position of a panel, call `SetPanelPos`.

Parameter List

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is contained in memory. `LoadPanel`, `NewPanel`, or `DuplicatePanel` will have returned this handle.
Height	integer	New height of the panel in pixels, excluding the panel frame and title bar. Valid values: 0 to 32,767.
width	integer	New width of the panel in pixels, excluding the panel frame. Valid values: 0 to 32,767.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetPlotAttribute

```
int status = SetPlotAttribute (int panelHandle, int controlID,
                           int plotHandle, int plotAttribute, ...);
```

Purpose

Sets the value of a graph plot attribute.

Parameters

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID you obtain from `NewCtrl` or `DuplicateCtrl`.
plotHandle	integer	Handle for a particular plot in the graph. You obtain the handle from one of the graph plotting functions.
plotAttribute	integer	Selects a particular graph plot attribute. Refer to Tables 3-59 and 3-60 in Chapter 3 for a complete listing of plot attributes.
attributeValue	depends on the attribute	New value of the plot attribute. Refer to Tables 3-59 and 3-60 in Chapter 3 type of for a complete listing of plot attributes.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetPrintAttribute

```
int status = SetPrintAttribute (int printAttribute, ...);
```

Purpose

Sets the value of the a particular print attribute.

LabWindows/CVI maintains only one copy of the print attributes you set with `SetPrintAttribute`. Thus, when you change a print attribute in one thread, the change affects printing functions you subsequently call in other threads.

Parameters

Input

Name	Type	Description
printAttribute	integer	A particular print attribute. Refer to Tables 3-69 and 3-70 in Chapter 3 for a complete listing of print attributes.
attributeValue	depends on the attribute	New value of the print attribute. Refer to Tables 3-69 and 3-70 in Chapter 3 for a complete listing of print attribute values.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetSleepPolicy

```
int status = SetSleepPolicy (int sleepPolicy);
```

Purpose

When the User Interface Library checks for an event from the operating system, it can put your program in the background, to sleep, for a specified period of time. This gives other applications more processor time, but your program might run more slowly.

Use `SetSleepPolicy` to set the amount your program sleeps when LabWindows/CVI checks 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 an adjustment is necessary, try the different settings and observe the resulting behavior.

LabWindows/CVI maintains a separate sleep policy for each thread of your program.

Parameter

Input

Name	Type	Description
sleepPolicy	integer	Degree to which your program periodically sleeps each time the User Interface Library checks for events from the operating system.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

Parameter Discussion

Sleep policy can be one of the following:

Defined Constant	Value	Description
`VAL_SLEEP_NONE`	1	Never be put to sleep.
`VAL_SLEEP_SOME`	2	Be put to sleep for a short period.
`VAL_SLEEP_MORE`	3	Be put to sleep for a longer period. The default value.

SetSystemAttribute

```
int status = SetSystemAttribute (int systemAttribute, ...);
```

Purpose

Sets the value of a particular system attribute.

Parameters

Input

Name	Type	Description
systemAttribute	integer	Refer to Table 3-66 in Chapter 3 for a complete listing of all system attributes.
attributeValue	depends on the attribute	New value of the system attribute. Refer to Table 3-66 in Chapter 3 for a complete listing of all system attributes.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetSystemPopupsAttribute

```
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.

The system popup attributes apply to all of the LabWindows/CVI popup panels, such as the graph pop-ups, the message pop-ups, and the file select popup on Windows 3.1 and UNIX.

The system popup attributes do not apply to the dialog boxes that are native to the Windows operating systems, such as the Windows 95/NT file dialog box. `FileSelectPopup`, `MultiFileSelectPopup`, and `DirSelectPopup` use the Windows 95/NT file dialog box.

LabWindows/CVI maintains only one copy of the attributes you set with `SetSystemPopupsAttributes`.

Parameters

Input

Name	Type	Description
popupAttribute	integer	`ATTR_MOVABLE` or `ATTR_SYSTEM_MENU_VISIBLE`
attributeValue	depends on the attribute	New value of the attribute. Refer to Table 3-2 in Chapter 3 for valid values associated with these attributes.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetTraceAttribute

```
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

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.
controlID	integer	Defined constant, located in the `.uir` header file, that you assign to the control in the User Interface Editor, or the ID that you obtain from `NewCtrl` or `DuplicateCtrl`.
traceNumber	integer	Identifies a strip chart trace. Can be from `1` to the number of strip chart traces. You set the number of strip chart traces in the User Interface Editor or through `SetCtrlAttribute.`
traceAttribute	integer	Selects a particular strip chart trace attribute. Refer to Table 3-59 in Chapter 3 for a complete listing of trace attributes.
attributeValue	integer	New value of the trace attribute.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SetWaitCursor

```
int status = SetWaitCursor (int waitCursorState);
```

Purpose

Specifies the state of the wait cursor. If you activate the wait cursor, LabWindows/CVI overrides all other cursor styles to display the wait cursor.

LabWindows/CVI maintains the state of the wait cursor independently for each thread. Thus, a call to `SetWaitCursor` in one thread does *not* cause the wait cursor to appear while a panel from another thread is the active panel.

Parameter

Input

Name	Type	Description
waitCursorState	integer	Specifies the state of the wait cursor. `1` = wait cursor active `0` = wait cursor inactive

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

SuspendTimerCallbacks

```
int status = SuspendTimerCallbacks (void);
```

Purpose

Stops all timer callbacks until you call `ResumeTimerCallbacks`.

`SuspendTimerCallbacks` does not alter the ongoing schedule of timer intervals. It only inhibits LabWindows/CVI from calling callback functions.

`SuspendTimerCallbacks` affects only timer controls you loaded or created in the active thread.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

See Also

`ResumeTimerCallbacks`, `ResetTimer`

UnRegisterWinMsgCallback

```
int status = UnRegisterWinMsgCallback (int callbackID);
```

☞ **Note** ***Only the Windows versions of LabWindows/CVI supports*** `UnRegisterWinMsgCallback`**.**

Purpose

Unregisters a Windows message callback function you registered with `RegisterWinMsgCallback`.

Parameter

Input

Name	Type	Description
callbackID	integer	ID you obtain from the **callbackID** parameter of `RegisterWinMsgCallback`.

Return Value

Name	Type	Description
status	integer	`1` = Successfully unregistered the callback function `0` = Failed to unregister the callback function

See Also

`RegisterWinMsgCallback`

ValidatePanel

```
int status = ValidatePanel (int panelHandle, int *valid);
```

Purpose

Verifies that the value of each numeric control for which `ATTR_CHECK_RANGE` is `VAL_NOTIFY` is within its valid range. You set the valid range in the User Interface Editor or by calling `SetCtrlAttribute` with `ATTR_MAX_VALUE` and `ATTR_MIN_VALUE`.

If a control is out of range, a red outline appears around it prompting the end-user to enter a new value. When the control has the input focus, a pop-up appears indicating the range and default value of the control.

Note ***LabWindows/CVI calls*** `ValidatePanel` ***when it is preparing to generate a commit event in response to user input on a control that is in Validate mode. LabWindows/CVI generates the commit event only if all controls are in range.***

Parameter

Input

Name	Type	Description
panelHandle	integer	Specifier for a particular panel that is currently in memory. You obtain this handle from `LoadPanel`, `NewPanel`, or `DuplicatePanel`.

Output

Name	Type	Description
valid	integer	Boolean value that indicates whether each control in `VAL_NOTIFY` mode is within its valid range. `0` = one or more controls are out of range `1` = all controls are in range

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

WaveformGraphPopup

```
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 graph control in a dialog box.

`WaveformGraphPopup` scales the values in **yArray** according to **yGain** and **yOffset**. It scales the x-axis timebase according to **initialX** and **xIncrement**. It computes each point in the plot as follows:

$$x_i = (i \times \mathbf{xIncrement}) + \mathbf{initialX}$$

$$y_i = (\mathrm{wfm}_i \times \mathbf{yGain}) + \mathbf{yOffset}$$

where *i* is the index of the point in the waveform array.

Parameters

Input

Name	Type	Description
title	string	Title to display on the dialog box.
yArray	void *	Array that contains the values to plot along the y-axis. The data type must be of the type you specify in **yDataType**.
numberOfPoints	integer	Number of points to plot. This value determines the number of points to plot even if the number of elements in **xArray** is greater than the **numberOfPoints.**
yDataType	integer	Specifies the data type of the **yArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.
yGain	double-precision	Specifies the gain to be applied to the waveform (**yArray**) data.
yOffset	double-precision	Specifies a constant offset to be added to the waveform (**yArray**) data.
initialX	double-precision	Specifies the initial value for the x-axis.
xIncrement	double-precision	Specifies the increment along the x-axis for each new point.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

XGraphPopup

```
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 of a graph control in a dialog box.

Parameters

Input

Name	Type	Description
title	string	Title to display in the dialog box.
xArray	void *	Array that contains the values to plot along the x-axis. The data type must be of the type you specify in **xDataType.**
numberOfPoints	integer	Number of points to plot. This value determines the number of points to plot even if the number of elements in **xArray** is greater than the **numberOfPoints**.
xDataType	integer	Specifies the data type of the **xArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

XYGraphPopup

```
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 on a graph control in a dialog box.

Parameters

Input

Name	Type	Description
title	string	Title to display in the dialog box.
xArray	void *	Array that contains the values to plot along the x-axis. The data type must be of the type you specify in **xDataType**.
yArray	void *	Array that contains the values to plot along the y-axis. The data type must be of the type you specify in **yDataType**.
numberOfPoints	integer	Number of points to plot. This value determines the number of points to plot even if the number of elements in **xArray** is greater than the **numberOfPoints**.
xDataType	integer	Specifies the data type of the **xArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.
yDataType	integer	Specifies the data type of the **yArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

YGraphPopup

```
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 on a graph control in a dialog box.

Parameters

Input

Name	Type	Description
title	string	Title to display on the dialog box.
yArray	void *	Array that contains the values to plot along the y-axis. The data type must be of the type you specify in **yDataType**.
numberOfPoints	integer	Number of points to plot. This value determines the number of points to plot even if the number of elements in **xArray** is greater than the **numberOfPoints**.
yDataType	integer	Specifies the data type of the **yArray**. Refer to Table 3-46 in Chapter 3 for a list of data types.

Return Value

Name	Type	Description
status	integer	Refer to Appendix A for error codes.

5 LabWindows/CVI Sample Programs

This chapter contains a list of the sample programs in LabWindows/CVI and a brief description of each. 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, refer to 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

Item Number	Project File Name	Program Description
1	`io.prj`	Standard I/O
2	`callback.prj`	Introduction to Callback Functions
3	`events.prj`	User Interface Events
4	`menus.prj`	Menus Controlled by Callback Functions
5	`graphs.prj`	Graphs
6	`chart.prj`	Strip Charts
7	`cursors.prj`	Graph Cursor
8	`popups.prj`	Pop-Up Panels
9	`listbox.prj`	Selection Lists
10	`panels.prj`	Child Windows
11	`timerctl.prj`	Timer Controls

Table 5-1. Sample Program Files (Continued)

Item Number	Project File Name	Program Description
12	`textbox.prj`	Text Boxes
13	`picture.prj`	Using Picture Controls
14	`build.prj`	Building a User Interface Programmatically
15	`getusrev.prj`	Programming with Event Loops
16	`keyfiltr.prj`	Handling Keyboard Input
17	`moustate.prj`	Getting the Mouse State
18	`listdelx.prj`	Colors in List Boxes
19	`2yaxis.prj`	Two y-axes on a Graph
20	`intgraph.prj`	Intensity Plots
21	`autostrp.prj`	Autoscaling the y-axis on a Strip Chart
22	`canvas.prj`	Canvas Controls
23	`canvsbmk.prj`	Canvas Benchmark
24	`drawpad.prj`	Using Canvas as Drawing Pad
25	`piedemo.prj`	Pie Chart
26	`imagedit.prj`	Changing Image Colors
27	`clipbord.prj`	Using System Clipboard
28	`scaling.prj`	Scaling Panel Contents

Using the Sample Programs

Each of these sample programs is 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 `SetCtrlAtrribute` 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 function panel for `InsertListItem`.

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 the 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 very simple program 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.

28. scaling.prj—Scaling Panel Contents

This sample program demonstrates how you can scale the contents of a panel when the user resizes it or you resize it programmatically.

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*** `GetUILErrorString` ***function can convert an error code number into a message string.***

Table A-1. User Interface Library Error Codes

Code	Error Message
`-1`	The Interface Manager could not be opened.
`-2`	The system font could not be loaded.
`-3`	The operation attempted cannot be performed while a pop-up menu is active.
`-4`	Panel, pop-up, menu bar, or plot ID is invalid.
`-5`	Attempted to position panel at an invalid location.
`-6`	Attempted to make an inoperable control the active control.
`-7`	The operation requires that a panel be loaded.
`-8`	The operation requires that a pop-up menu be active.
`-9`	The operation requires that a menu bar be loaded.
`-10`	The control is not the type expected by the function.
`-11`	Invalid menu item ID.
`-12`	Out of memory.
`-13`	Invalid control ID.
`-14`	Value is invalid or out of range.
`-15`	File is not a User Interface file or has been corrupted.

Table A-1. User Interface Library Error Codes (Continued)

Code	Error Message
`-16`	File format is out-of-date.
`-17`	PCX image is corrupted or incompatible with current display type.
`-18`	No user event possible in current configuration.
`-19`	Unable to open UIR file.
`-20`	Error reading UIR file.
`-21`	Error writing UIR file.
`-22`	Error closing UIR file.
`-23`	Panel state file has invalid format.
`-24`	Invalid panel ID or menu bar ID in resource file.
`-25`	Error occurred during hardcopy output.
`-26`	Invalid default directory specified in `FileSelectPopup` function.
`-27`	Operation is invalid for specified object.
`-28`	Unable to find specified string in menu.
`-29`	Palette menu items can only be added to the end of the menu.
`-30`	Too many menus in the menu bar.
`-31`	Separators cannot have checkmarks.
`-32`	Separators cannot have submenus.
`-33`	The menu item must be a separator.
`-34`	The menu item cannot be a separator.
`-35`	The menu item already has a submenu.
`-36`	The menu item does not have a submenu.
`-37`	The control ID passed must be a menu ID, a menu item ID, or NULL.
`-38`	The control ID passed must be a menu ID, or a menu item ID.
`-39`	The control ID passed was not a submenu ID.
`-40`	The control ID passed was not a valid ID.
`-41`	The ID is not a menu bar ID.

Table A-1. User Interface Library Error Codes (Continued)

Code	Error Message
`-42`	The ID is not a panel ID.
`-43`	This operation cannot be performed while this pop-up panel is active.
`-44`	This control/panel/menu/ does not have the specified attribute.
`-45`	The control type passed was not a valid type.
`-46`	The attribute passed is invalid.
`-47`	The fill option must be set to fill above or fill below to paint ring slide's fill color.
`-48`	The fill option must be set to fill above or fill below to paint numeric slide's fill color.
`-49`	The control passed is not a ring slide.
`-50`	The control passed is not a numeric slide.
`-51`	The control passed is not a ring slide with inc/dec arrows.
`-52`	The control passed is not a numeric slide with inc/dec arrows.
`-53`	The data type passed in is not a valid data type for the control.
`-54`	The attribute passed is not valid for the data type of the control.
`-55`	The index passed is out of range.
`-56`	There are no items in the list control.
`-57`	The buffer passed was to small for the operation.
`-58`	The control does not have a value.
`-59`	The value passed is not in the list control.
`-60`	The control passed must be a list control.
`-61`	The control passed must be a list control or a binary switch.
`-62`	The data type of the control passed must be set to a string.
`-63`	That attribute is not a settable attribute.
`-64`	The value passed is not a valid mode for this control.
`-65`	A NULL pointer was passed when a non-NULL pointer was expected.

Table A-1. User Interface Library Error Codes (Continued)

Code	Error Message
`-66`	The text background color on a menu ring cannot be set or gotten.
`-67`	The ring control passed must be one of the menu ring styles.
`-68`	Text cannot be colored transparent.
`-69`	A value cannot be converted to the specified data type.
`-70`	Invalid tab order position for control.
`-71`	The tab order position of an indicator-only control cannot be set.
`-72`	Invalid number.
`-73`	There is no menu bar installed for the panel.
`-74`	The control passed is not a text box.
`-75`	Invalid scroll mode for chart.
`-76`	Invalid image type for picture.
`-77`	The attribute is valid for child panels only. Some attributes of top-level panels are determined by the host operating system.
`-78`	The list control passed is not in check mode.
`-79`	The control values could not be completely loaded into the panel because the panel has changed.
`-80`	Maximum value must be greater than minimum value.
`-81`	Graph does not have that many cursors.
`-82`	Invalid plot.
`-83`	New cursor position is outside plot area.
`-84`	The length of the string exceeds the limit.
`-85`	The specified callback function does not have the required prototype.
`-86`	The specified callback function is not a known function.
`-87`	Graph cannot be in this mode without cursors.
`-88`	Invalid axis scaling mode for chart.
`-89`	The font passed is not in font table.

Table A-1. User Interface Library Error Codes (Continued)

Code	Error Message
`-90`	The attribute value passed is not valid.
`-91`	Too many files are open.
`-92`	Unexpectedly reached end of file.
`-93`	Input/Output error.
`-94`	File not found.
`-95`	File access permission denied.
`-96`	File access is not enabled.
`-97`	Disk is full.
`-98`	File already exists.
`-99`	File already open.
`-100`	Badly formed pathname.
`-101`	File is damaged.
`-102`	The format of the resource file is too old to read.
`-103`	File is corrupted.
`-104`	The operation could not be performed.
`-105`	The control passed is not a ring knob, dial, or gauge.
`-106`	The control passed is not a numeric knob, dial, or gauge.
`-107`	The count passed is out of range.
`-108`	The keycode is not valid.
`-109`	The picture control has no image.
`-110`	Panel background cannot be colored transparent.
`-111`	Title background cannot be colored transparent.
`-112`	Not enough memory for printing.
`-113`	The shortcut key passed is reserved.
`-114`	The format of the file is newer than this version of LabWindows/CVI.
`-115`	System printing error.

Table A-1. User Interface Library Error Codes (Continued)

Code	Error Message
`-116`	Driver printing error.
`-117`	The deferred callback queue is full.
`-118`	The mouse cursor passed is invalid.
`-119`	Printing functions are not reentrant.
`-120`	Out of Windows GDI space.
`-121`	The panel must be visible.
`-122`	The control must be visible.
`-123`	The attribute is not valid for the type of plot.
`-124`	Intensity plots cannot use transparent colors.
`-125`	Color is invalid.
`-126`	The specified callback function differs only by a leading underscore from another function or variable. Change one of the names for proper linking.
`-127`	Bitmap is invalid.
`-128`	There is no image in the control.
`-129`	Specified operation can be performed only in the thread in which the top-level panel was created.
`-130`	Specified panel was not found in the `.tui` file.
`-131`	Specified menu bar was not found in the `.tui` file.
`-132`	Specified control style was not found in the `.tui` file.
`-133`	A tag or value is missing in the `.tui` file.

B

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

E-Mail Support (Currently USA Only)

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

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name ____

Company ____

Address ____

____

Fax (___) _________________ Phone (___) ________________________________

Computer brand_____________ Model ____________________ Processor______________________

Operating system (include version number) ________________________________

Clock speed ______MHz RAM _____MB Display adapter ____________________

Mouse ___yes ___no Other adapters installed________________________________

Hard disk capacity _____MB Brand____

Instruments used ____

____

National Instruments hardware product model______________ Revision ______________________

Configuration ____

National Instruments software product____________________ Version _______________________

Configuration ____

The problem is: ____

____

____

____

____

List any error messages: ____

____

____

The following steps reproduce the problem: ____________________________________

____

____

____

____

LabWindows/CVI Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

Hardware revision ____________________

Interrupt level of hardware ____________________

DMA channels of hardware ____________________

Base I/O address of hardware ____________________

Programming choice ____________________

National Instruments software ____________________

Other boards in system ____________________

Base I/O address of other boards ____________________

DMA channels of other boards ____________________

Interrupt level of other boards ____________________

Other Products

Computer make and model ____________________

Microprocessor ____________________

Clock frequency or speed ____________________

Type of video board installed ____________________

Operating system version ____________________

Operating system mode ____________________

Programming language ____________________

Programming language version ____________________

Other boards in system ____________________

Base I/O address of other boards ____________________

DMA channels of other boards ____________________

Interrupt level of other boards ____________________

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *LabWindows/CVI User Interface Reference Manual*

Edition Date: February 1998

Part Number: 320683D-01

Please comment on the completeness, clarity, and organization of the manual.

____
____
____
____
____
____
____

If you find errors in the manual, please record the page numbers and describe the errors.

____
____
____
____
____
____
____

Thank you for your help.

Name ____

Title ____

Company ____

Address ____

____

E-Mail Address ____

Phone (___) ____________________________ Fax (___) __________________________

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
	micro-	10^{-6}

B

binary switch	Control that selects between two states: on and off.
bitmap	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	Arbitrary drawing surface to display text, shapes, and bitmap images.
CodeBuilder	LabWindows/CVI feature that creates source 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	Message box that allows you to confirm an action before it is taken.
control	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.

F

file select pop-up panel	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	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 commit events.

I

immediate command	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	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	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	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	Predefined pop-up panel for displaying a message.
ms	Milliseconds.

N

normal control	Control that can be operated by the user and changed programmatically. Normal controls generate all events except commit events.
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	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	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	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*.
plot	Consists of a curve, a point, a geometric shape, or a text string.
point	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**.
pop-up panel	Panel that pops up, accepts user input, and then disappears.

prompt pop-up panel	Predefined pop-up panel for requesting input from the user.
pull-down menu	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

rect	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.
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	Cross-bar of the slide control that points to the currently selected item.
string control	See *numeric/string control*.
strip chart control	Graph that displays graphical data as one or more traces in real time.

T

text boxes	Display a window of text.
text controls	Display a string of text.
timer control	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	Environment where you create resource files for a user interface.

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