Controls

Preliminary Release

February, 1997

C H A P T E R 1

Controls

Contents

Controls

This chapter describes controls available through the Mac OS Toolbox that a user manipulates in windows, dialog boxes, and alert boxes. As the name of this class of elements implies, most controls offer the user some measure of control over system and application activity through choices and the ability to set parameters. Some of these controls are indicators used to convey information about application process duration and system capacity to the user. Some of these controls are primitives used as parts of other controls or as background. The appearance and behavior of these controls, indicator controls, and primitive controls is standardized.

Controls comprise buttons—including push buttons, radio buttons, checkboxes, bevel buttons, and popup menu buttons—large sliders and tick marks, little arrows, disclosure triangles, scrolling lists, list boxes, scroll bars, edit text fields, tabs, placards, and image wells.

Primitive controls include primary and secondary group boxes, list box frames, edit text frames, separator lines, window headers, and modeless dialog fill.

Indicator controls include large progress indicators (both determinate and indeterminate ones) and asynchronous arrows.

Controls are graphic objects that the user can manipulate with the mouse causing instant actions or producing audible results. Because of their appearance and behavior, controls enhance the user's sense of direct manipulation and influence on the configuration and future behavior of the system and application. Users manipulate controls that configure settings to modify future actions. Controls also allow users to make choices or assign parameters in a range. Controls display existing choices so that they are visible to users.

When an operation requires more than one object or when it needs additional information before it executes, the Mac OS interface uses dialog boxes to gather information and alert boxes to convey it. These dialog boxes and alert boxes contain controls that users manipulate to provide the additional input. Controls are also found in windows. Controls provide users with familiar tools and formats for responding to the computer's need for information.

Indicator Controls are graphic objects that provide users with information about the application activity and system capacity. For example, when an application is performing a task that entails duration, the application might display a progress indicator to symbolically convey to the user the transacted and remaining time required for the activity. Indicators might also represent available memory or disk capacity.

Not to be confused with dialog primitives, **primitive controls** are graphic objects used with other controls, for example, to delimit them visually. For example, a list box frame is used to surround a scrolling list field and separator lines are used to delineate sections of the content area horizontally or vertically.

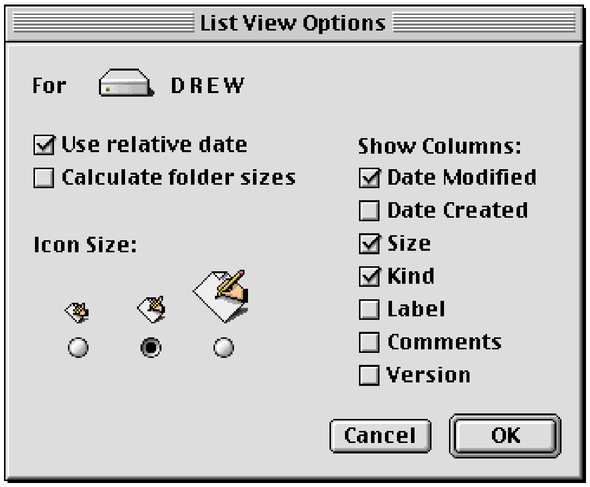
Controls

This section describes controls that the user manipulates. It includes descriptions of the appearance and behavior of push buttons, radio buttons, checkboxes, bevel buttons having various uses, large sliders and tick marks, little arrows, disclosure triangles, scroll bars, scrolling lists, and edit text fields. It briefly describes popup menu buttons, which are described completely in "Menus" (page 46).

Push Buttons

A **push button** is a rounded rectangle that is named with text. For example, Figure 1-1 shows some typical push buttons in the List View Options dialog box.

Figure 1-1 Push buttons in a dialog box


List View Options

For DREW

[✓] Use relative date
[ ] Calculate folder sizes

Icon Size: ○ [●] ○

Show Columns:
[✓] Date Modified
[ ] Date Created
[✓] Size
[✓] Kind
[ ] Label
[ ] Comments
[ ] Version

Clicking a push button initiates the action described by the button's name. Push buttons usually perform instantaneous actions, such as completing operations defined by a dialog box or acknowledging an error message. Standard push button height is 20 pixels. A push button's width is sized to fit the name it surrounds. The standard size for the OK and Cancel buttons is 20 pixels by 58 pixels.

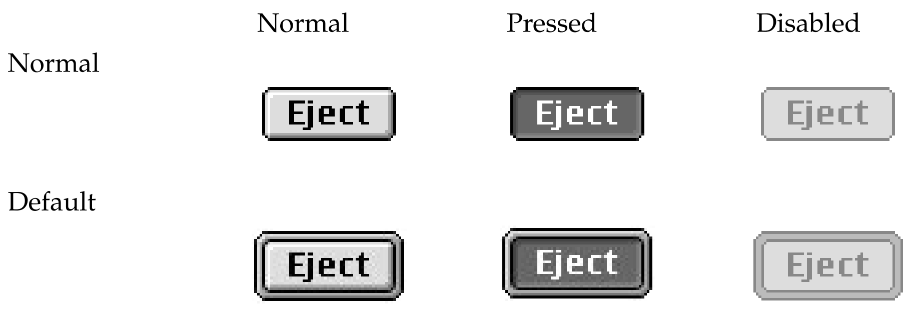
A push button has three states: normal, pressed, and disabled. The normal state of a push button indicates that the function the button controls is available but the user has not yet enacted it by clicking on the button. This is the usual state of a button when the dialog box or window containing it is first displayed.

A button is depicted in its pressed state when the user clicks on it to enact its function. When the user clicks a push button, the button highlights (inverts) to give visual feedback to the user that indicates which item has been clicked.

A button is displayed in its disabled state when the function it represents is not available or meaningful within the current context or when the button is drawn in a background window.

Dialog boxes and windows containing more than one button should always include a default button. Define as the default button the one that the user is most likely to click in response to an inquiry or information submittal request the dialog box or window presents to the user. The default button should also be the one that effects the least destructive results. You set an attribute or property of the button-drawing function to define a push button so that it is drawn as the default one. The default push button has a ring whose appearance is coordinated with the state of the button. Figure 1-2 shows the three states of standard and default push button.

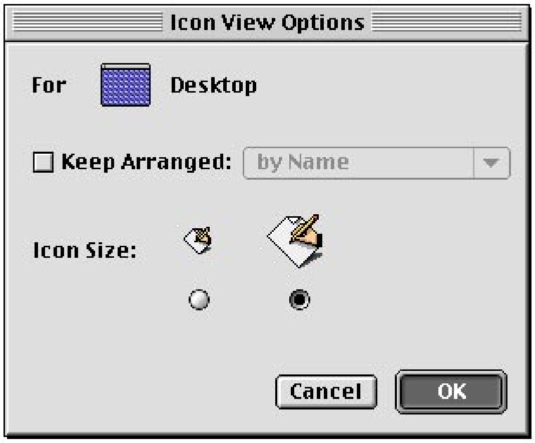
Figure 1-2 Normal, pressed, and disabled states of standard and default push



When the user clicks a push button, the button highlights to give visual feedback to the user that indicates which item has been clicked. All alert boxes and modal dialog boxes that use the ModalDialog procedure exhibit this behavior. If you implement your own controlling mechanism for dialog boxes or alert boxes, be sure to include this behavior.

For push buttons that are activated by using a keyboard sequence, the Dialog Manager highlights the button for eight ticks, which is long enough for the user to see that the keyboard event has taken effect. (You must highlight the Cancel button when the user presses Command-period or the Escape key; the Dialog Manager does not handle these events.) If the user presses the mouse button while the pointer is over a push button, the button remains highlighted until the user releases the mouse button or moves the pointer away from the push button. The push button tracks the mouse movement as long as the user keeps the mouse button depressed. If the user moves the pointer back over the push button, it is highlighted. If the user releases the mouse button while the pointer is not over the push button, nothing happens. Figure 1-3 shows a push button that is highlighted to provide feedback.

Figure 1-3 A highlighted push button



The **default push button** should be the button that represents the action that the user is most likely to perform if that action isn't potentially dangerous. If your application has set the `setDefaultDialogItem` property for a push button, then the push button is drawn with the default ring style. See the *Inside Macintosh: Toolbox Essentials Reference* for information on the push-button drawing function.

When the user presses the Enter key or the Return key, your application should respond as if the user clicked the default button.

Avoid using a default push button if the most likely action is dangerous—for example, if it causes a loss of user data. Don't use a default push button if you use the Return key in text entry boxes. Having two behaviors for one key can confuse users and make the interface less predictable.

For example, you should not use a default push button if the dialog box contains a scrollable, editable text field into which the user can type multiple lines of text. To do so would cause the default action to occur when the user pressed the Return key to position the cursor on the next line to enter more text.

When there is no default push button, pressing Return or Enter has no effect; the user must explicitly click a button. This guideline protects users from accidentally damaging their work by pressing Return or Enter. You can consider using a safe default push button, such as Cancel.

In addition to the positive action push button or buttons, it's a good idea to include a Cancel push button. This push button returns the computer to the state it was in before the dialog box appeared. It allows the user to retreat from an action, as if to say to your application "I didn't mean it." Always map the keyboard equivalent Command-period and the Esc (Escape) key to the Cancel push button. These keyboard equivalents, along with Return and Enter, are accelerator keys; they serve the purpose of letting the user quickly express certain responses to a dialog box or an alert box. In general, it's not a good idea to assign other keyboard equivalents to push buttons. If you find it useful to assign keyboard equivalents to some push buttons that are used very often in your application, be sure to follow the"Keyboard Equivalents" (page 106) guidelines in "Menus".

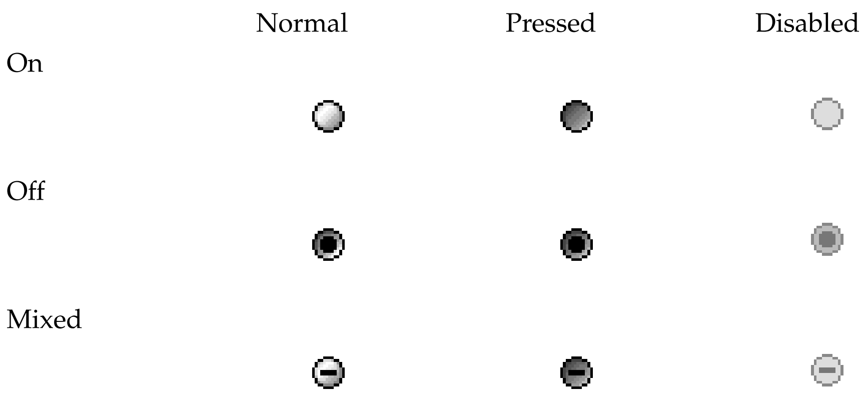
For information about implementing these behaviors for push buttons, see *Inside Macintosh: Toolbox Essentials* and *Inside Macintosh: Toolbox Essentials Reference*.

Radio Buttons

A radio button is a Mac OS control that displays a setting, either on or off, and is part of a group in which only one button can be on at a time. They occur in sets and are called radio buttons because they act like the buttons—not the tuning knobs—on a car radio. Just as the user can have only one radio button setting in effect at a time to listen to one radio station, only one radio button control within a set of them can be active at a time. Radio buttons are mutually exclusive in terms of effect. The active setting has a dot in the middle of the button. Clicking one button in a group turns off whichever button was on before. Radio buttons never initiate an action.

Radio buttons appear differently in their normal, pressed, and disabled states depending on whether they are on, off, or in mixed mode. Figure 1-4 shows how radio buttons look in their various states.

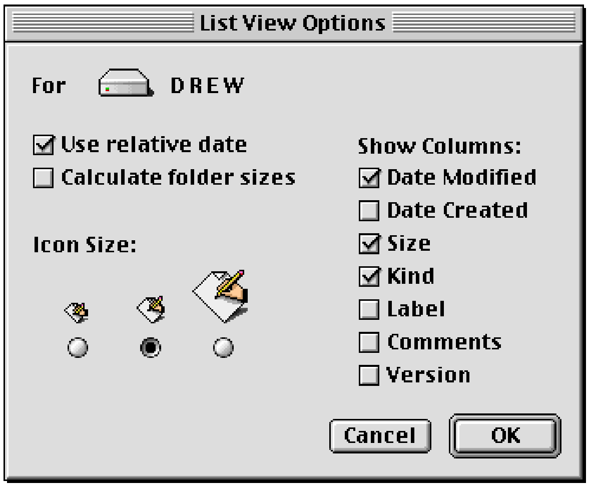
Figure 1-4 Radio buttons in normal, pressed, and disabled modes



A set of radio buttons should contain from two to approximately seven items. Some sets could be slightly larger, but you must always have at least two radio buttons in each set. Each group of radio buttons usually has a label that identifies the kind of choices the group contains. Usually, each button has a label that identifies what it does. A label can be a few words or a phrase. A set of radio buttons always has the same set of choices. It is *never* dynamic, changing contents depending on the context. To activate the button, the user can click the button itself or the text that identifies the choice it represents.

Radio buttons represent related choices, but not necessarily opposite ones. For example, a set of radio buttons may offer the user choices of different sounds, such as alert sounds the system can make, or it may offer the user choices of different sizes for icon display. Figure 1-5 shows a dialog box that uses radio buttons to offer icon size choices to the user.

Figure 1-5 Radio buttons for selecting the icon size



List View Options

For DREW

✓ Use relative date

☐ Calculate folder sizes

Icon Size: ○ [●] ○

Show Columns:

✓ Date Modified

☐ Date Created

✓ Size

✓ Kind

☐ Label

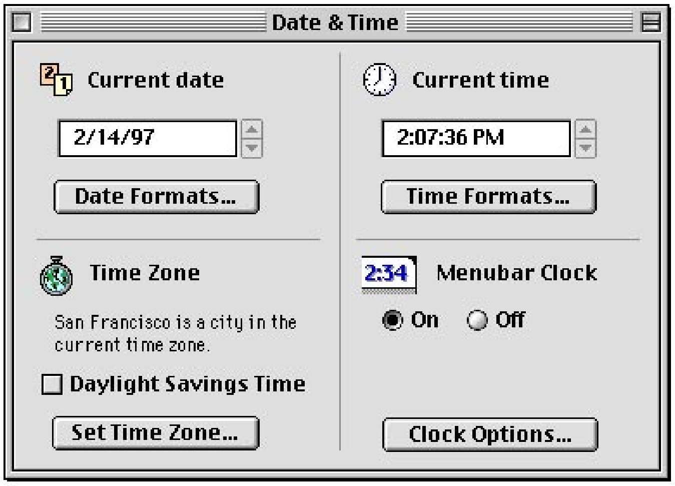
☐ Comments

☐ Version

If more than one group of radio buttons is visible at one time, the groups need to be visually separate from each other. You should design your dialog box to allow enough visual space to clearly delineate multiple sets of radio buttons. Although it does not contain multiple sets of radio buttons, Figure 1-5 follows the concept in using visual space to distinguish sets of radio buttons and checkboxes from one another.

Sometimes it’s useful to draw a line around a group of radio buttons to separate it from other elements in a dialog box. For this purpose, you can use the separator line control primitive described in “Separator Lines” (page 48).

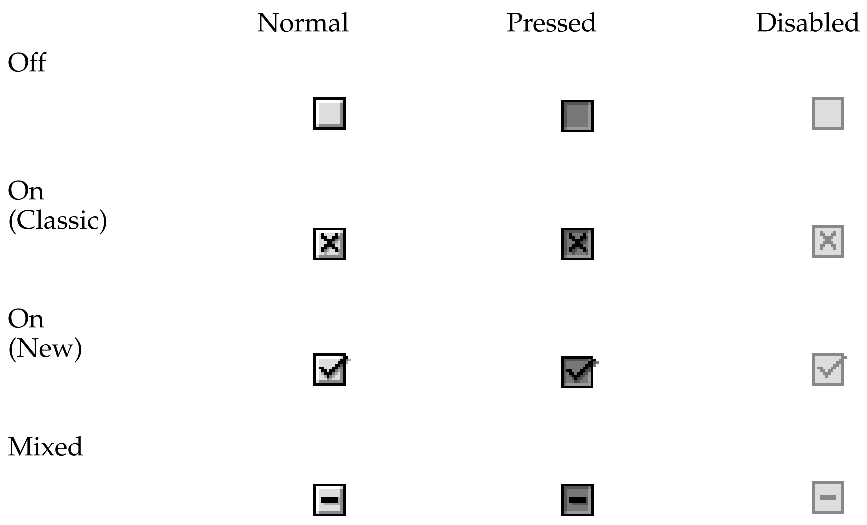
Figure 1-6 Separating a set of radio buttons using a separator line control primitive



Checkboxes

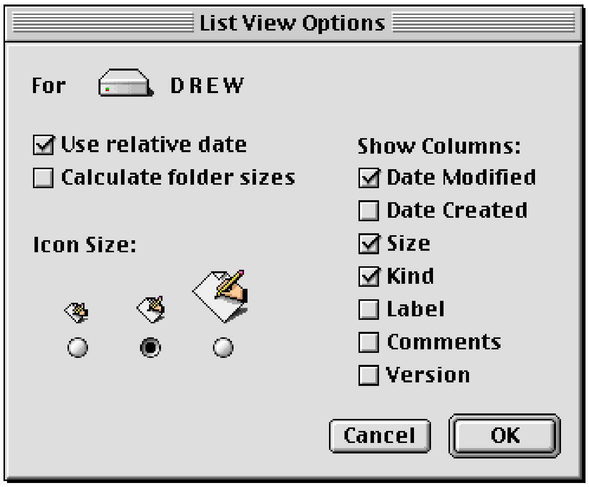
Checkboxes, like radio buttons, provide alternative choices for users. A **checkbox** is a square with a text label next to it. The user clicks the checkbox to select or deselect it. When an option is off, the box is empty. The Mac OS Toolbox look and feel for checkboxes includes two variations for marking the checkbox in its selected state. When an option is on, a checkmark is used to indicate that the checkbox is the selected one. The checkmark is used for all cases in which it is culturally acceptable. The classic checkmark is used instead of the new, standard Mac OS Toolbox one for those cultural regions in which the new one holds connotation different from its intended use for the checkbox control. Figure 1-7 includes both of these checkmarks. Figure 1-7 illustrates how checkboxes look in their normal, pressed, and disabled modes for their on, off, and mixed states. It

Figure 1-7 Checkboxes in normal, pressed, and disabled modes



Checkmarks act like toggle switches, meaning that the setting for each checkbox is either off or on. Checkboxes are independent of each other, even when they offer related options. Use checkboxes to indicate one or more options that must be either off or on. Any number of checkboxes can be on or off at the same time. Figure 1-8 shows a set of checkboxes with three checkboxes selected to specify the kind of information to be displayed.

Figure 1-8 A set of checkboxes with multiple concurrent selections



List View Options

For DREW

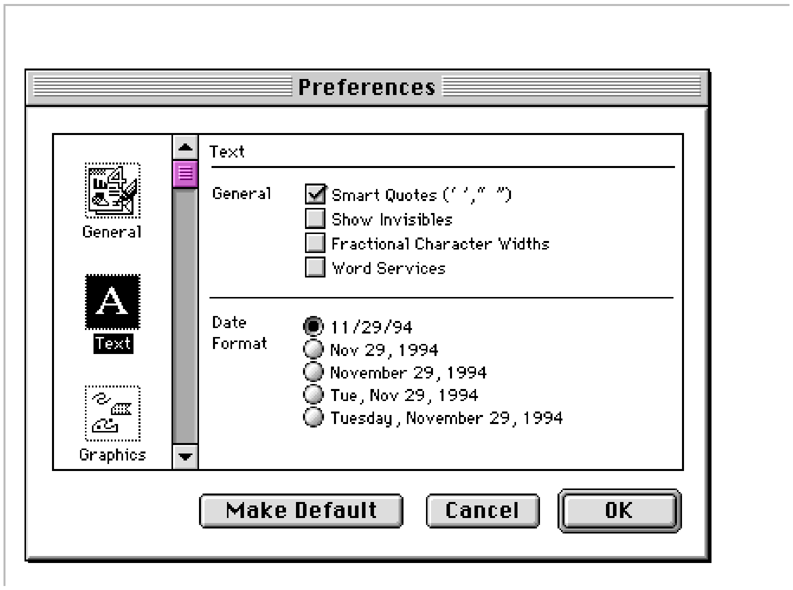
[✓ Use relative date]
Calculate folder sizes

Icon Size: ○ [●] ○

Show Columns:
[✓ Date Modified]
Date Created
[✓ Size]
[✓ Kind]
Label
Comments
Version

You can have one checkbox or as many as you need. It's a good idea to group sets of checkboxes that are related and to separate the groups from other groups of checkboxes and radio buttons. For example, Figure 1-9 shows an application's Preferences control panel that contains a set of checkboxes isolated from a set of radio buttons using a separator line. See "Separator Lines"(page 48) for more information. Note that this preliminary figure does not show the grayscale background that is standard for the new Mac OS Toolbox.

Figure 1-9 Checkboxes isolated using a separator line



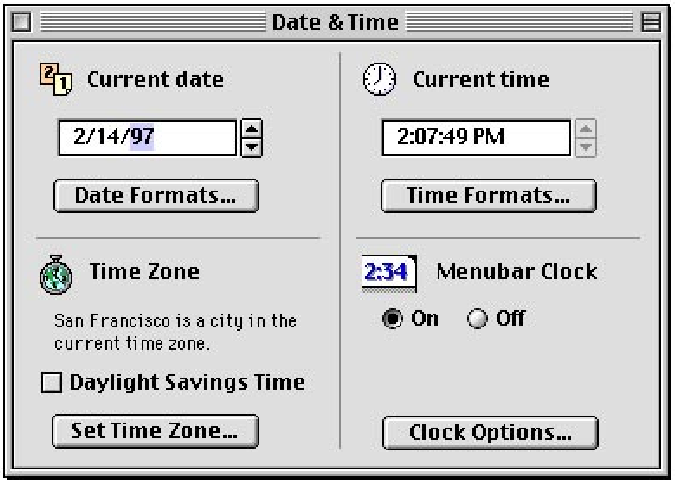
Text

General
- [✓] Smart Quotes (‘ ’,“ ”)
- [ ] Show Invisibles
- [ ] Fractional Character Widths
- [ ] Word Services

Date Format
- ● 11/29/94
- ○ Nov 29, 1994
- ○ November 29, 1994
- ○ Tue, Nov 29, 1994
- ○ Tuesday, November 29, 1994

Each checkbox has a label. It can be very difficult to label the option in an unambiguous way. The label should imply two clearly opposite states. For example, a dialog box for opening files could include a checkbox labeled Read-Only that provides the option to open a file in a read-only format. The clearly opposite state, when the option is off, is to open files that the user can read *and* write (or make changes to). Figure 1-10 shows a dialog box with a checkbox that, when selected, sets the system clock to adjust for daylight savings time.

Figure 1-10 A single checkbox in a dialog box



If you can't find a label for the checkbox that clearly implies its opposite state, you might be better off using radio buttons. With radio buttons, you can use two labels, thereby clarifying the states. It's sometimes tempting to use a checkbox because one item takes up less space than two. However, the resulting item may be ambiguous and thus difficult for your users to understand.

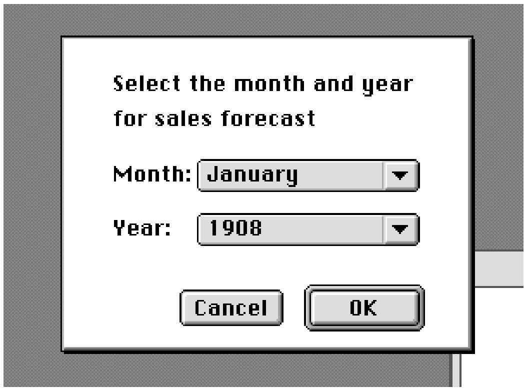
When you use one checkbox to provide two options, it makes the user think explicitly about what the significance of the option is. The user must click the checkbox or its label text to enable the option. In this way, you can emphasize the visible choice.

Popup Buttons

A popup button is a single control that is depicted as a button with two distinct parts. Even though the button is one control, the user can click each part separately. The left side of the button contains the label whose text identifies the currently selected menu item, or value. The right side of the button depicts a downward-pointing triangle. The representation of the button as if it were two parts indicates to the user that both the label and triangle are selectable.

When the user presses down on the mouse button with the pointer over the popup button, a menu of choices is displayed directly below the popup button. The open menu of choices includes the value that currently appears in the popup button. A checkmark appears next to this value in the open menu. For complete information about the popup button, see "Menus" (page 46). Figure 1-11 shows two popup buttons in their normal states.

Figure 1-11 Popup buttons in their normal state

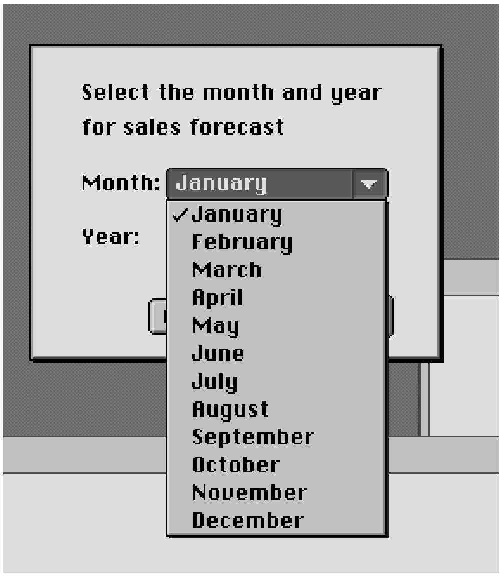

Select the month and year for sales forecast

Month: January

Year: 1908

Figure 1-12 shows the Month popup button with the menu displayed after the user has clicked on either selectable portion of the button.

Figure 1-12 A popup button with its menu displayed



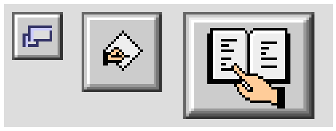
Bevel Buttons

A bevel button is a standard square-shaped button control whose content area can present text, a graphical object such as a picture or an icon, or both. For bevel buttons that contain text, you can specify how the text is justified and aligned. For bevel buttons that contain both text and an image, you can also specify placement of the text in relation to the image. Bevel button text is system-orientation aware, meaning that the text is aligned according to the system orientation. For buttons that contain both text and an image, the image will be correctly placed according to the system orientation if the image and text are on the same horizontal plane.

A bevel button mimics the behavior of other button types. You can attach a menu to a bevel button, in which case the bevel button takes on the behavior of a popup button. The popup bevel button supports both standard and sticky menu behavior. A bevel button can behave like a standard push button; in this case, the button pops back up after the user clicks on it. Bevel buttons can also behave in sets as radio buttons and as checkboxes.

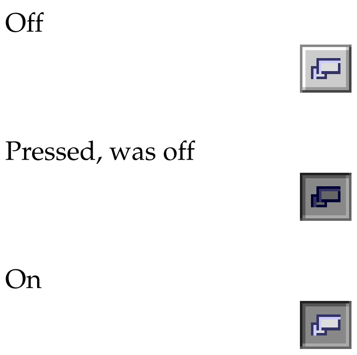
You can choose from among three sizes of bevels——small, normal, and large—for any size button you create. The button is outlined with a bevel. The small bevel is two pixels wide; the normal bevel is three pixels wide; and the large bevel is four pixels wide. Figure 1-13 shows examples of these bevels.

Figure 1-13 Small, normal, and large bevel buttons

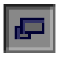


Regardless of its size, a bevel button is characterized by seven states, depicted in Figure 1-14:

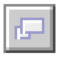
Figure 1-14 Bevel button states



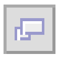
Pressed, was on



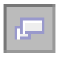
Mixed



Disabled, off



Disabled, on

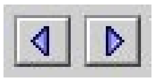


To understand the mixed state, assume that a set of bevel buttons is used as part of a word processing application to allow the user to apply style to selected ranges of text. The set includes bevel buttons for boldfacing, italicizing, and underling text. The mixed state occurs when the selected range of text includes some characters that are boldfaced and some characters that are italicized.

Bevel Buttons as Push Buttons

A bevel button can be used as a push button, in which case it takes on a push button's behavior while retaining the bevel button appearance. Depending on your requirements, you can use any number of bevel buttons serving as push buttons in your dialog box or window, just as you can use any number of standard push buttons together. Figure 1-15 shows two bevel buttons

Figure 1-15 A pair of bevel buttons used as push buttons



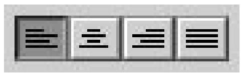
When the user clicks a bevel button that behaves as a push button, it initiates the action the button represents. Visually, the button returns to its off state. If the user rolls the cursor off the button at any time while holding down the mouse button, the bevel button reverts to its up state.

Bevel buttons used as push buttons have the same behavior as standard bevel buttons, which is described in"Push Button Behavior" (page 6).

Bevel Buttons as Radio Buttons

A bevel button can be used as a radio button, in which case it takes on a radio button's behavior while retaining the bevel button appearance. The user can toggle the button on and off: one click turns the button on, another turns it off. Bevel buttons acting as radio buttons are used in sets, just as standard radio buttons are. For example, you might want to use a set of them to provide a text justification tool in a tool bar. For a bevel button used as a radio button, only one button is effective at any given time. The behavior for standard radio buttons, described in "Radio Buttons" (page 4), applies to radio bevel buttons. Figure 1-16 shows a group of bevel buttons serving as radio buttons.

Figure 1-16 A set of bevel buttons used as radio buttons



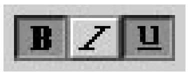
Bevel Buttons as Checkboxes

A bevel button can be used as a checkbox, in which case it takes on the checkboxes behavior, described in "Checkboxes" (page 11), while retaining the bevel button appearance. The user can toggle the button on and off: one click turns the button on, another turns it off. You can use checkbox bevel buttons singly or in sets. For example, you could use a set of checkbox bevel buttons as a text styling tool in a toolbar.

Unlike radio buttons, any of the checkbox bevel buttons in a set can be switched on at any time while others are also on. Their actions are not mutually exclusive. After a mouse up occurs—that is, after the user releases the mouse

button—the button changes from the pressed state to the on state. See "Bevel Buttons" (page 17) and Figure 1-14 (page 18) for bevel button states.
Figure 1-17 shows a group of bevel buttons used as checkboxes in a text styling tool. The buttons labeled B for boldface and U for underline are depressed in the on state because the selected range of text contains both boldfaced and italicized characters.

Figure 1-17 A set of bevel buttons used as checkboxes



Bevel Buttons as Popup Buttons

A bevel button can be used as a popup button. A popup bevel button has a double set of behaviors. In one case, it behaves just like a standard popup button. For example, the user clicks on the bevel button while holding down the mouse button to display a popup menu directly below or to the right of the button. The popup button menu itself can exhibit two behaviors: standard or sticky menu. The menu behavior depends on how long the user holds down the mouse button. In its capacity as a standard menu, the menu closes immediately after the user releases the mouse button. In its capacity as a sticky menu, after the user clicks on the button, the menu remains displayed— this behavior is useful for tool palettes. Figure 1-18 shows a popup bevel button that serves only as a popup.

Figure 1-18 A popup bevel button used only as a popup

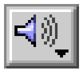
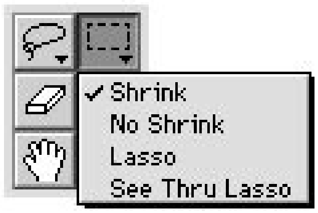


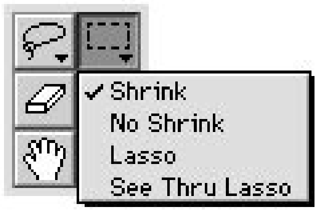
Figure 1-19 shows an example of a popup bevel button that can not only display a popup menu, but also maintain its selected state when the user releases the mouse button.

Figure 1-19 A popup bevel button



In the other case for a popup bevel button, the standard popup button behavior is effective, but so also is some of the behavior of radio buttons and checkbox buttons. For example, to simply turn on the function represented by the button, the user clicks once on the bevel button while releasing the mouse button. This initiates the function without displaying the popup menu. The standard behavior occurs if the user clicks on the bevel button while holding down the mouse button—a popup menu is displayed.

Figure 1-20 A popup bevel button



When you use the Mac OS Toolbox to create a popup bevel button, you can specify that it contain either small or large arrows. Figure 1-21 shows both sets.

Figure 1-21 Small and large arrows for popup bevel buttons

Button Names

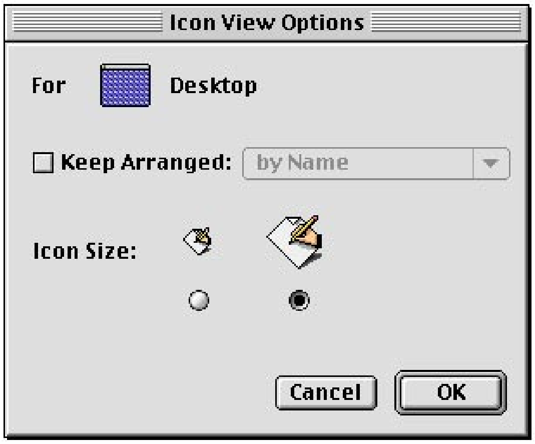
Whenever possible, name a push button or a radio button with a verb that describes the action the button performs. Try to limit button names to one word. In any case, you should never use more than three words for a button name. Use the caps/lowercase style of capitalization for button names. In general, this means that you capitalize the first letter of every word except articles *(a, an, the)*, coordinating conjunctions (for example, *and, or*), and prepositions of three or fewer letters. You also capitalize the first letter of the first and last words of the name. Because button names should seldom be more than two words, almost all words in button names should be capitalized. The specific rules for this type of capitalization appear in detail in the *Apple Publications Style Guide*.

Button names usually appear in the large system font. If a button is not available, it appears in its disabled state. On a black-and-white monitor, a disabled button and its name are dithered to 50 percent gray. On color systems, disabled items appear in true gray.

Buttons usually cause instant actions, described by the name of the button. Occasionally they require more information before enacting an effect. If a button displays another dialog box, use the ellipsis character in the button name to indicate this to the user. (Do not include the ellipsis character if the dialog box appears only to ask users to confirm their actions.) The most appropriate situation for a second dialog box to appear would be when a modeless dialog box needed additional information on a limited basis to complete an action. See the section “Stacking Modal Dialog Boxes” (page 137) in “Dialog Boxes” for a discussion of the dangers of dialog boxes that generate more dialog boxes.

A user typically reads the text in a dialog box until it becomes familiar and then relies on visual cues, such as button names or positions, to respond. Names such as Save, Quit, and Erase Disk allow users to identify and click the correct button quickly. These words are often more clear and precise than names such as OK, Yes, and No. If the action can't be condensed into a word or two, OK and Cancel or Yes and No may serve the purpose. If you use these generic words, be sure to phrase the wording in the dialog box so that the action the button initiates is clear. Figure 1-22 shows a dialog box with appropriate OK and Cancel buttons.

Figure 1-22 A dialog box with OK and Cancel buttons

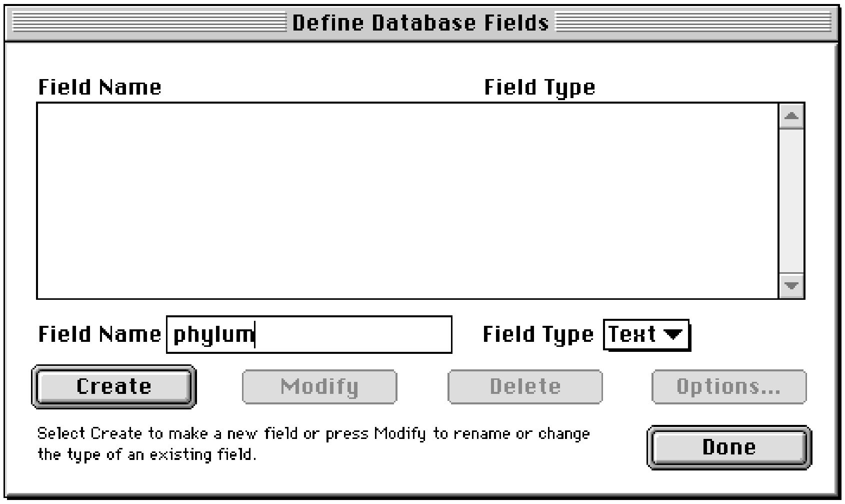


Use Cancel for the button that closes the alert box or dialog box and returns the application to the state it was in before the alert box or dialog box appeared. Cancel means "dismiss the operation I started, with no side effects." It does not mean "I've read this dialog box" or "stop what's going on regardless."

Ideally, you should never put your users in a situation in which they can't return to the state that existed before an operation began (and before the dialog box appeared). But if it can't be avoided, use OK or Stop, depending on the situation, instead of Cancel. Sometimes it is more appropriate to use the word Done instead of OK for the name of a button that closes the alert box or dialog box and that accepts any changes made while the dialog box is displayed.

Figure 1-23 shows a dialog box that illustrates this guideline. In this dialog box, the user creates, modifies, or deletes fields and then dismisses the dialog box, so the Done button means "I have finished editing fields and want to close the dialog box." Note that this preliminary figure does not show the grayscale background that is standard for the new Mac OS Toolbox.

Figure 1-23 A dialog box with a Done button instead of an OK button

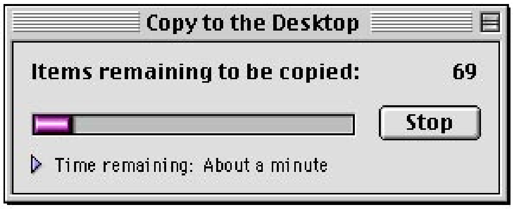


Select Create to make a new field or press Modify to rename or change the type of an existing field.

This dialog box uses Done because clicking the Done button maintains any changes that were made subsequent to the display of the dialog box. If the button were named OK, the user might confuse it with the Add Field button, which accepts changes but doesn't close the dialog box and therefore allows the user to make other changes. The Done button is most often used in dialog boxes in which the user can define more than one of an item, for example, field names, without closing the dialog box. In these situations it is often unreasonably difficult to return the user to the state that existed before the operation began, so there is no Cancel button.

Use Stop for a button that halts an operation midstream while accepting the possible side effects. Stop may leave the results of a partially complete task intact, whereas Cancel always returns the computer to its previous state. It's appropriate to change the button name in the middle of the operation from Cancel to Stop if you can determine when it's no longer possible to cancel. The dialog box shown in Figure 1-24 uses a Stop button to stop further copying of data without deleting the data that has already been successfully transferred.

Figure 1-24 A progress indicator that uses a Stop button



Items remaining to be copied: 69

Time remaining: About a minute

In an alert box that requires confirmation, use a word that describes the result of accepting the message in the dialog box. For example, if a dialog box says "Revert to the last saved version of this document?" name the button Revert rather than OK.

A modal dialog box usually cuts the user off from the task. That is, the user can't see the area of the document that changes when choices are made in the dialog box until dismissing the dialog box. Once the area becomes visible because the user dismisses the dialog box, the user sees whether the changes are the desired ones. If the changes aren't appropriate, then the user has to repeat the entire operation. To give better feedback to the user, you need to provide a way for the user to see what the changes will be. Therefore, any selection made in a modal dialog box should immediately update the document contents, or you should provide a sample area in the dialog box that reflects the changes that the user's choices will make. In the case of immediate document updating, the OK button means "accept this change" and the Cancel button means "undo all changes done by this dialog box."

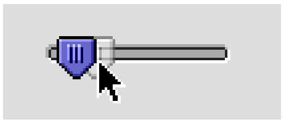
Some applications use an Apply button to approximate the behavior of immediately updating the document or using a sample area. This method confuses the meaning of OK and Cancel and is not recommended. If you must implement modal dialog boxes with an Apply button, you need to include a Cancel button. When there is an Apply button, the Cancel button undoes the results of the Apply operation and dismisses the dialog box. The OK button dismisses the dialog box and applies the settings made in the dialog box, even if the Apply button wasn't clicked. The user must always be able to undo any actions caused by the dialog box.

Large Sliders and Tick Marks

A **slider** displays the range of values, magnitude, or position of something in the application or system. An indicator notes the current setting. For Tempo, only the large slider is supported.

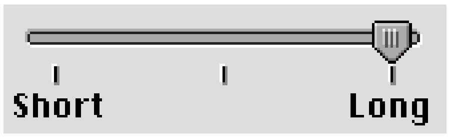
The large slider is 16 pixels tall, which is a fixed value. You can specify the length of the slider when you create it using the Mac OS Toolbox. The large slider has a slider thumb indicator that can point in any direction or it can be nondirectional. Sliders support live dragging. The user can drag the indicator thumb across a horizontal slider or up and down a vertical one to alter the value of the slider or to change what is displayed. Like scroll bars, sliders drag a ghost image of the indicator along with them. The ghost region is a copy of the indicator region offset by a certain amount based on where the user dragged the indicator. Like scroll bars, sliders support live feedback. Figure 1-25 shows a large slider with a downward pointing indicator shadowed by a ghost indicator.

Figure 1-25 Large slider and directional indicator



Built into the slider control are tick marks that you can use to depict incremental values. Tick marks are optional; you can specify whether or not they should be drawn with the slider. Sliders can be horizontal or vertical. If you use tick marks, they are drawn appropriately for the direction of the slider. Figure 1-26 shows a horizontal slider using vertical tick marks.

Figure 1-26 A horizontal slider

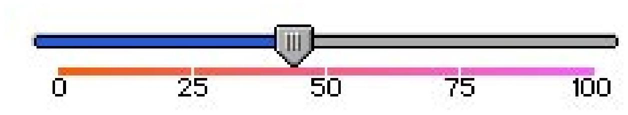


You can design and implement your own sliders as necessary for your application. When you design your sliders, be sure to include meaningful labels that indicate to users the range and direction of the slider.

Give users clues about the direction in which the indicator moves and how that relates to the control. While tick marks indicate the directions in which the indicator moves—whether up and down or right and left—they do not depict in which direction the value increments. For instance, most people assume that moving an indicator up a vertical slider means increasing the value of the setting. However, this assumption could be clarified easily with graphics or words.

Figure 1-27 shows an example of a horizontal slider that uses incremental numbering to indicate an increase as the user moves the indicator to the right. This type of explicit labeling substantially improves the comprehensibility of the graphical interface.

Figure 1-27 A slider with direction information



Make sure that you don't use a scroll bar when you really mean to use a slider. Use scroll bars only for representing the relative position of the visible portion of a document and in scrolling lists. Typically a scroll bar represents the amount of data in a document, and the scroll box represents the relative position of the window over the length of the document. Using a scroll box to change a setting confuses the meaning of the element and makes the interface inconsistent. See "Scroll Bars" (page 115) in "Windows" for details.

Little Arrows

The control consisting of two arrows pointing in opposite directions is commonly called **little arrows.** Little arrows provide the user with a means of increasing or decreasing values in a series.

Figure 1-28 shows an example of little arrows that allow the user to increment or decrement the date.

Figure 1-28 The little arrows

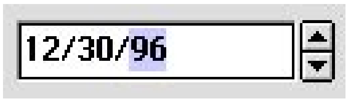
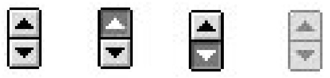


Figure 1-29 shows the little arrows control in its normal state, with the up arrow depressed, with the down arrow depressed, and in its disabled state.

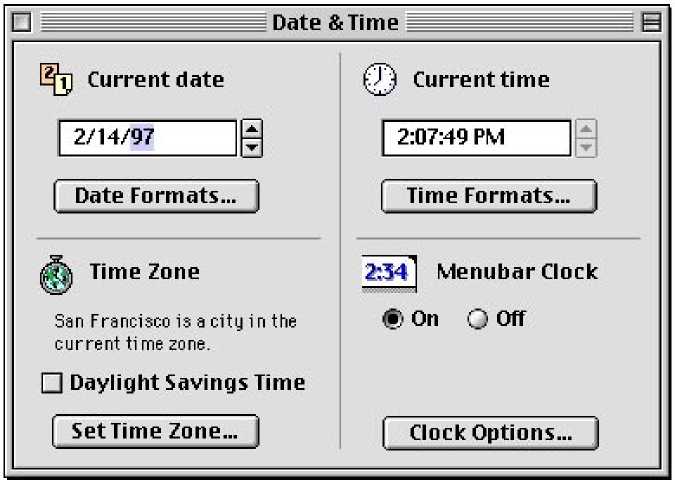
Figure 1-29 Little arrows control in various states



Traditionally, little arrows have been used in the Mac OS Date and Time control panel. You can use the little arrows control in your own control panels to ensure that they implement the Mac OS Toolbox look and feel.

Figure 1-30 shows the little arrows control used for the current date display The little arrows control has a label that specifies the content to which it relates. The numerical or textual value appears in a box, which is often a type-in box so the user can type in a value instead of using the little arrows. When the user clicks one arrow, the value changes by a unit of 1. If the user presses the arrow, the value increases or decreases until the user releases the mouse button. While the user clicks or presses the arrow, it is highlighted to provide feedback to the user. The unit of change depends on the content. For example, if the content area displays years, the increment is one year at a time, as shown in Figure 1-30.

Figure 1-30 Little arrows with content-dependent increment



Current date: 2/14/97

Current time: 2:07:49 PM

Time Zone

San Francisco is a city in the current time zone.

☐ Daylight Savings Time

Menubar Clock: [✓ On] Off

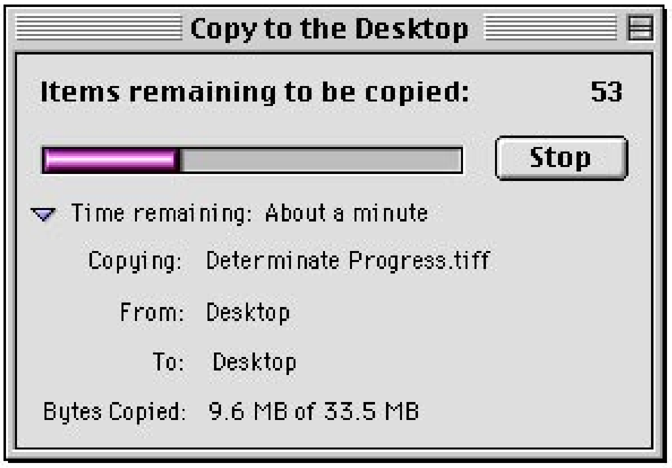
If possible, give some indication what the user can expect by using the up arrow and the down arrow. For example, in Figure 1-30 it may not be obvious which direction the year would increment using either arrow. Clicking the up arrow might change the year from 97 to 98 or 97 to 96.

The little arrows control works best with numbers in cases in which it is obvious that the up arrow means 1 more than the current value and the down arrow means 1 less than the current value.

Disclosure Triangles

The **disclosure triangle** is a control that is used for displaying or "disclosing" information that is additional to or an elaboration of primary information in a dialog box.

Figure 1-31 The disclosure triangle revealing additional information



Copy to the Desktop

Items remaining to be copied: 53

Stop

Time remaining: About a minute

Copying: Determinate Progress.tiff

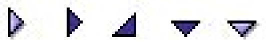
From: Desktop

To: Desktop

Bytes Copied: 9.6 MB of 33.5 MB

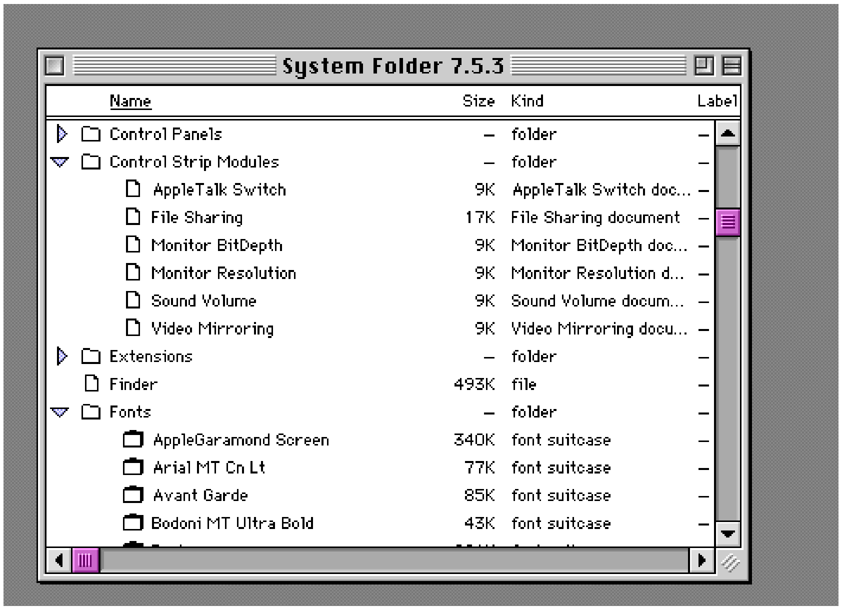
The disclosure triangle control implements the turning triangle widget, as shown in all of its positions and states in Figure 1-32, not the hierarchical list.

Figure 1-32 The disclosure triangle in its various positions and states



The disclosure triangle is used in the Finder. Users see it when they choose to display the contents of their file system in a list view. The triangle appears next to folders that contain documents or other folders. The user clicks the triangle to reveal a list of the contents of the folder without actually opening it. The triangle then rotates to point downward. This change in position indicates to the user that the folder's contents are listed. Figure 1-33 shows the disclosure triangle in both positions. Note that this preliminary figure does not show the grayscale background that is standard for the new Mac OS Toolbox.

Figure 1-33 Disclosure triangle control



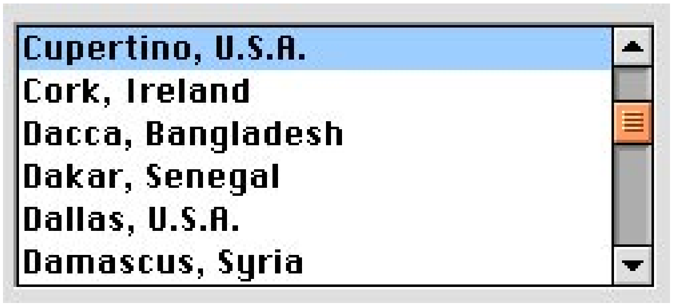
System Folder 7.5.3

Name	Size	Kind	Label
▷ Control Panels	–	folder	–
▽ Control Strip Modules	–	folder	–
AppleTalk Switch	9K	AppleTalk Switch doc...	–
File Sharing	17K	File Sharing document	–
Monitor BitDepth	9K	Monitor BitDepth doc...	–
Monitor Resolution	9K	Monitor Resolution d...	–
Sound Volume	9K	Sound Volume docum...	–
Video Mirroring	9K	Video Mirroring docu...	–
▷ Extensions	–	folder	–
Finder	493K	file	–
▽ Fonts	–	folder	–
AppleGaramond Screen	340K	font suitcase	–
Arial MT Cn Lt	77K	font suitcase	–
Avant Garde	85K	font suitcase	–
Bodoni MT Ultra Bold	43K	font suitcase	–

List Boxes and Scrolling Lists

A scrolling list is a combination of two elements. One part of the scrolling list is a list of items, such as a list of document names displayed in a list box. The list can contain as many items as necessary. You can use the list box element to implement a scrolling list box with a white background; the list box provides a complete solution that includes a list box frame. The size of the list box that displays the list often depends on the amount of space available in the context. Figure 1-34 shows an example of a list box, including a list box frame, used for a scrolling list. Note that this preliminary figure does not show the grayscale background that is standard for the new Mac OS Toolbox.

Figure 1-34 A list box



The other component to a scrolling list is a scroll bar, which allows the user to look at more items in the list than are currently visible. “Scroll Bars” (page 35) provides additional information. To specify the name and an icon, if appropriate, that labels the list box, you use an auxiliary resource.

When you create the list of items in a scrolling list, you may find text that is too long to fit in the list. When this is the case, it’s best to eliminate text in the middle of the name and insert ellipsis points there, preserving the beginning and ending of the item’s name. Users often add version numbers to the end of their document or device names, so if you cut off the end of the text item, they lose that context and must guess which of the several item names that begin the same is the desired one.

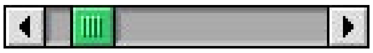
The user can click an item in the list to select it, or use multiple selection techniques such as the Shift-click combination to select more than one item. The user can also scroll through the list to peruse its contents without selecting anything. If the list contains folders, the user can use standard techniques to open them and see their contents. Users can also use the keyboard navigation techniques to select items in a scrolling list. See “Keyboard Navigation in Dialog Boxes” (page 142) in “Dialog Boxes” for more information.

Scrolling lists are not appropriate to use for providing choices in a limited range. Because the full range isn’t visible all at once in a scrolling list, it’s difficult for users to understand the scope of their choices. Sliders work very well for displaying a limited range of values and for letting users choose their preference in the range. See “Large Sliders and Tick Marks” (page 27) for information about sliders.

Scroll Bars

A **scroll bar** is a shaded gray rectangle that has an arrow in a box at each end. You can use scroll bars with windows as well as with scrolling lists. When used with windows, scroll bars allow the user to change which part of a document is shown in the window. Users can click in any area of the scroll box to move the displayed document content by a page. If the scroll bar is used for the Scrapbook or a similar application, clicking in the scroll box moves the displayed content an image at a time. Windows can have a horizontal scroll bar, a vertical scroll bar, or both. For complete information on the use of scroll bars with windows, see “Scrolling a Window” (page 112) in “Windows”. Figure 1-35 shows an example of a horizontal scroll bar.

Figure 1-35 A horizontal scroll bar

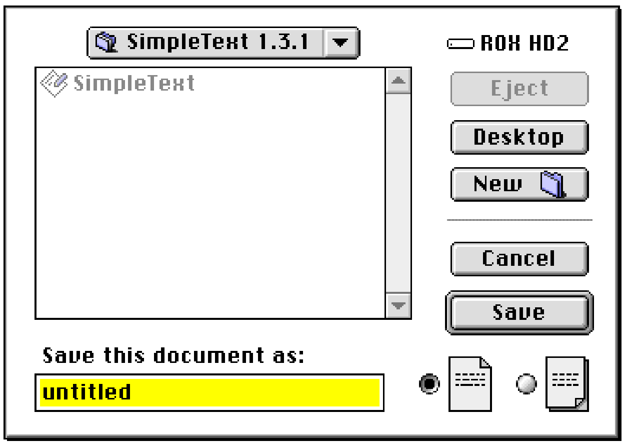


Edit Text Fields

The **edit text field** is a rectangular box in a dialog box in which the user enters text or modifies existing text. For example, in the Save As dialog box, the user types in the name of a document. The edit text field can be disabled and enabled. It allows keyboard filtering and supports password entry.

Figure 1-36 shows an edit text field in a dialog box that allows the user to save a document under another name. Note that this preliminary figure does not show the grayscale background that is standard for the new Mac OS Toolbox.

Figure 1-36 An edit text field



Save this document as:

untitled

If an application isn't primarily a text application, but does use text in fields, you may not need to provide the full text-editing capabilities. In Mac OS applications, the simplest way to implement text editing is to use TextEdit, or to use the Dialog Manager, which in turn uses TextEdit. You need to make sure that whatever level of text-editing capabilities you implement for edit text fields is upward compatible with the full text-editing capabilities. You should implement these editing capabilities:

- The user can select the whole field and type in a new value, delete text, select a substring of the field and replace it, and select a word by double-clicking.
- The user can choose Undo, Cut, Copy, Paste, and Clear, as described in "The Edit Menu" (page 92) in "Menus".

In addition, you can also implement intelligent cut and paste. (TextEdit does not provide this.) This capability is described in the section "Intelligent Cut and Paste" (page 226) in "Behaviors".

Even applications with only minimal text editing should perform appropriate edit checks. For example, if the only legitimate value for a field is a string of digits, the application should issue an alert message if the user types any nondigits. For example, the alert message might interrupt the user to remind him or her that the letters *l* and *o* can't be used in place of the numerals *1* and *0*. Alternatively, the application could wait until the user is through typing before checking the validity of a field's contents. In this case, the appropriate time to check the field is when the user clicks anywhere other than within the field or presses the Return, Enter, or Tab key.

Tabs

The tabs control provides a standard way for you to create multipane, multipaged information that is easy for your users to understand. Tabs consist of a grouped pane in which the content and interface objects are laid out. The pane is paged by a group of buttons that give the visual appearance of folder tabs.

The tabs control supports one row of tabs running along the top, as shown in Figure 1-37 and Figure 1-38. To specify the names and their icons that label the tabs, you use an auxiliary resource. Figure 1-37 shows the large tabs control with labels depicted in 12-point font.

Figure 1-37 Large tabs control with 12-point font

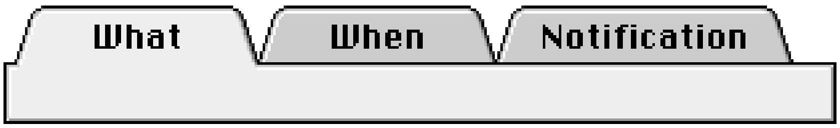
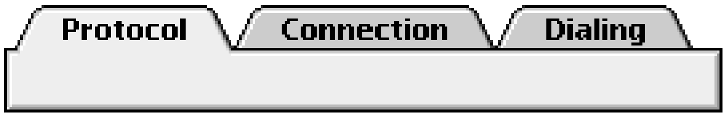


Figure 1-38 shows the small tabs control with 10-point font tab labels.

Figure 1-38 Small tab control with 10-point font tab labels



When the user presses down on the mouse button and the cursor is positioned over the clickable region of an inactive tab, the tab changes to the pressed state. If the user drags the cursor outside the clickable region of a tab while holding down the mouse button, the tab changes from the pressed state to the inactive state.

When the user releases the mouse button after pressing down on it, if the cursor is outside the clickable region of the tab, the tab remains as it was—no action is taken. However, if the cursor is still within the clickable region, the tab's content area changes to the active state. Figure 1-39 shows the active state of a tab.

Figure 1-39 A tab in its active state

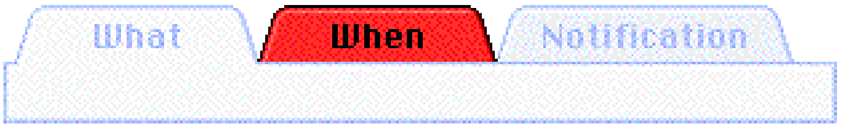
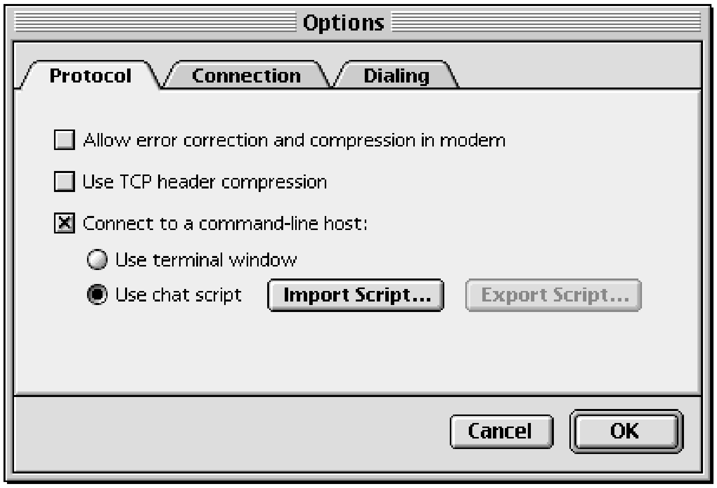


Figure 1-40 shows a tab control inside a document window. In this example, the control is tucked under the structure or content region by one pixel on both its left and right sides. Global controls are positioned below the tabbed region.

Figure 1-40 Tabs control with sides tucked under the content region



☐ Allow error correction and compression in modem

☐ Use TCP header compression

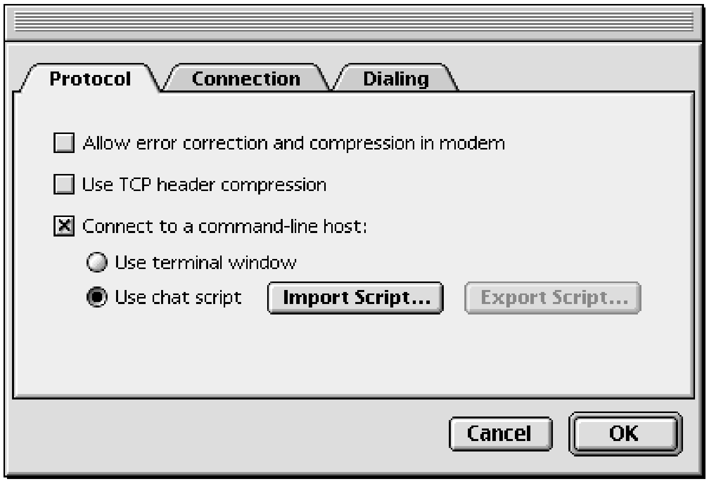
☒ Connect to a command-line host:

○ Use terminal window

● Use chat script

Figure 1-41 shows a tabs control used in a modal dialog. The left and right sides of the tabs control are inside the content region in this example.

Figure 1-41 Tab control with edges inside the content region



☐ Allow error correction and compression in modem

☐ Use TCP header compression

☒ Connect to a command-line host:

○ Use terminal window

● Use chat script

Figure 1-42 shows the tabs control with tucked edges. In this case, the tab region contains a scrollable content area.

Figure 1-42 Tabs control with tucked sides and a scrollable content area

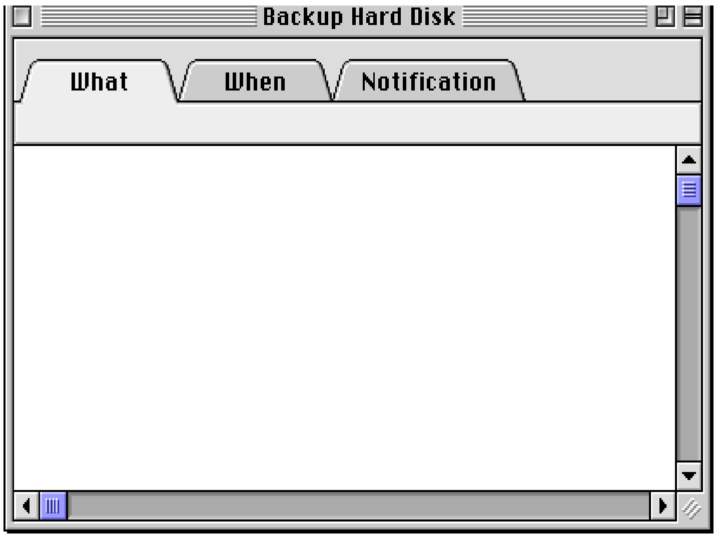
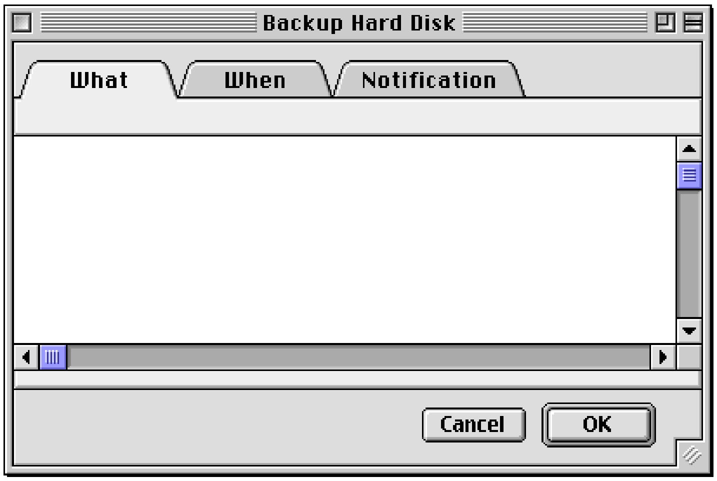


Figure 1-43 shows the tabs control with a scrollable content embedded in the tabbed region. This example also includes global buttons.

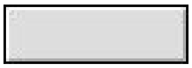
Figure 1-43 Tabs control with tucked sides, a scrollable content area, and global buttons



Placards

A placard is a control that you can use to raise elements up from their backgrounds or as background fill for a user control area. Figure 1-44 shows the placard control. A placard used as background denotes to the user that the area does not allow interaction. Placards have three states, normal, disabled, and pressed.

Figure 1-44 The placard control



You can use the placard control to implement a small panel to show information, such as the one often used at the bottom of a window to the left of the horizontal scroll bar to show the current line number. Figure 1-45 shows an example of a placard used to report disk efficiency.

Figure 1-45 A placard used to report information to the user

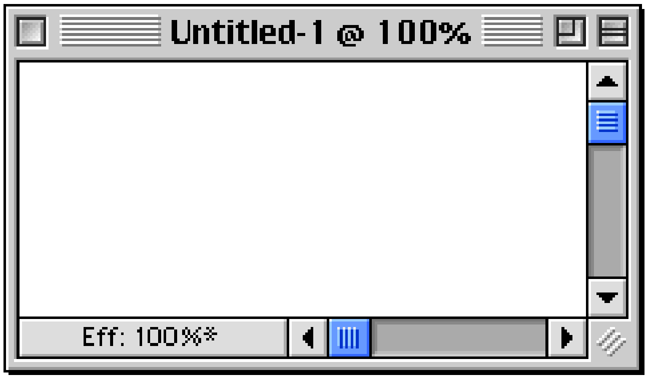
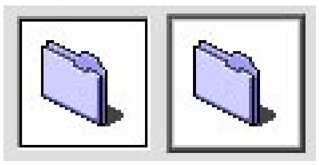


Image Wells

The image well control is used to contain any visual content other than text, for example, an icon or a picture. The image well line is two pixels wide. The image well has two states, enabled and disabled, as shown in Figure 1-46.

Figure 1-46 The image well in its enabled and disabled states



Primitive Controls

Controls primitives include primary and secondary group boxes, list box frames, edit text frames, separator lines, window headers, and modeless dialog fill. These controls can be used as parts of other controls or as background.

Group Boxes

The group box primitive control implements two types of group boxes—primary and secondary. Primary group boxes are used to associate groups of related items in a dialog box, distinguishing them from other items belonging to the dialog box. Primary group boxes are also used to indicate items in a dialog box that can be paged using a popup menu. The primary group box has an etched look.

All primary group box border lines are two pixels wide. A primary group box can have a title, but one is not required. You can use any of four titling options for the border of a group box: the group box can be untitled or it can have a text title, a popup menu title, or a check box title. Figure 1-47 shows an untitled primary group box.

Figure 1-47 An untitled primary group box

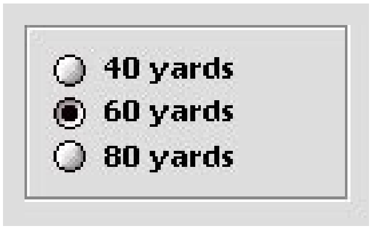


Figure 1-48 shows a titled primary text box using the large system font for its title text.

Figure 1-48 A titled primary group box

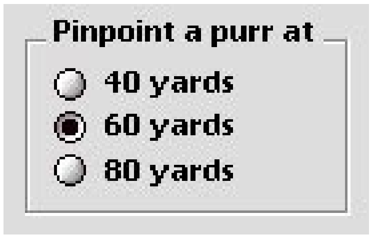


Figure 1-49 shows a primary text box using a popup menu title with its text depicted in the large system font.

Figure 1-49 A primary group box with a popup menu title

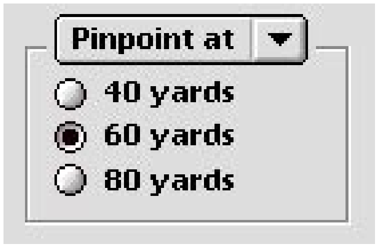
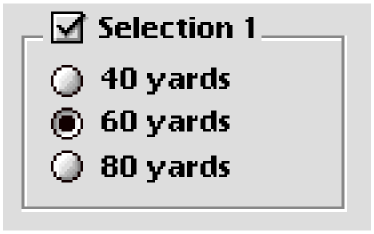


Figure 1-50 shows a primary group box with a checkbox title whose title text is depicted in the large system font.

Figure 1-50 A primary group box with a checkbox title

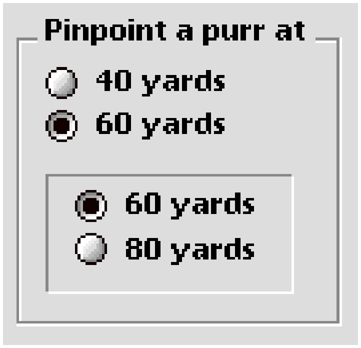


Secondary group boxes appear inside of primary ones. Their border is one pixel wide. Secondary group boxes are used primarily for nesting and grouping together subsidiary information.

A secondary group box has the same titling options as a primary group box. It can have a title, but one is not required. You can use any of four titling options for the border of a group box: the group box can be untitled or it can have a text title, a popup menu title, or a check box title.

Do not use secondary group boxes in place of primary ones. To do so would produce inconsistent dialog box appearances, serving only to confuse your user. You can embed other controls, such as radio buttons, within both the primary and the secondary group box controls.

Figure 1-51 A nested secondary group box with etched inset inside a primary one

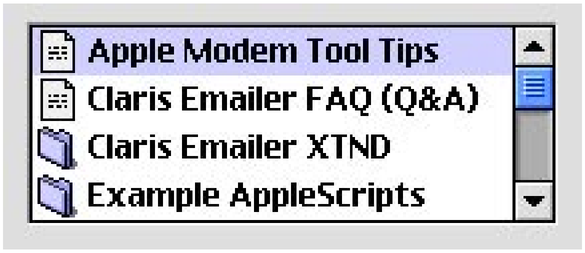


List Box Frames

The list box frame primitive control is a two-pixel wide rectangular frame whose inside lines share the outside lines of an active scrollbar. You can use the list box frame to surround a scrolling list field instead of using the list box control. The list box frame allows you to display pictures and other types of graphics in a scrolling list field.

Figure 1-52 shows an example of a list box, including a list box frame, used for a scrolling list.

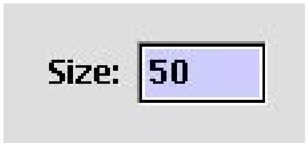
Figure 1-52 The list box frame



Edit Text Frames

The edit text frame primitive control is used to surround an editable text field. The text box frame is two pixels wide. It has two states, enabled and disabled. Figure 1-53 shows an example of it.

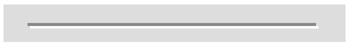
Figure 1-53 The edit text frame



Separator Lines

The separator line primitive control is used in dialog boxes to delineate horizontal or vertical regions of the content area. The single-state separator line is two pixels wide. For horizontal lines, the top pixel creates the line itself and the bottom pixel gives it the engraved effect. For vertical lines the left pixel creates the line and the right pixel gives it the engraved effect. Figure 1-54 shows the separator line.

Figure 1-54 The separator line

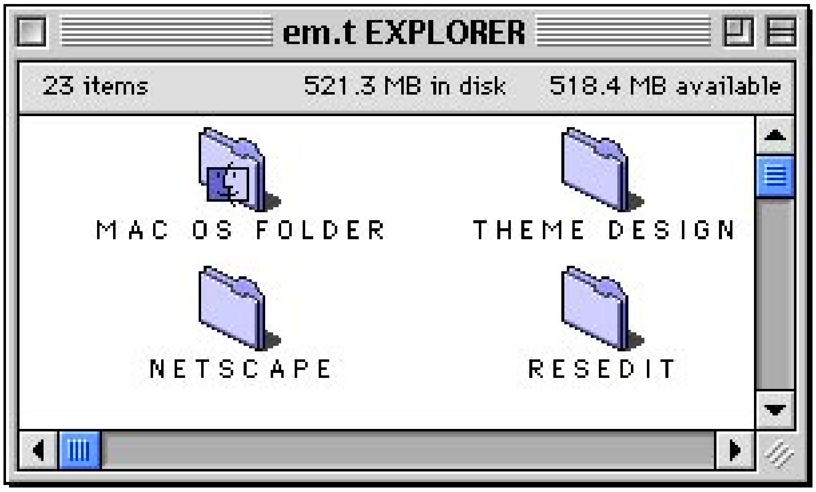


Window Header

The window header primitive control is used to distinguish the information section of a window from its content region. There are two types of window headers, one that is used in list view and another that is used in icon view.

The icon view header is a beveled rectangle whose outside lines share the same space as the inside lines of the document window and the scroll bar arrows. Figure 1-55 shows an example of the icon view header.

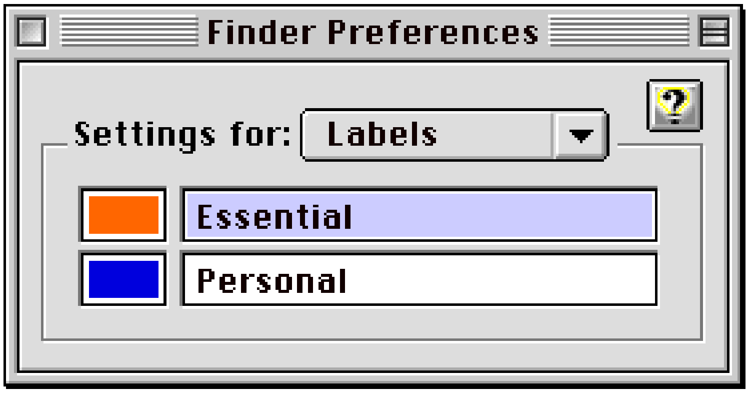
Figure 1-55 A Finder window using the icon view header



Modeless Dialog Fill

The modeless dialog fill control primitive is used as fill for the content region of a movable modeless dialog. It is a two-pixel wide rectangle whose outside lines share the inside lines of the document window. It has two states, foreground and background. Figure 1-56 shows a modeless dialog fill in a dialog box; this dialog box does not follow standard design but rather is used solely to illustrate use of the modeless dialog fill.

Figure 1-56 The modeless dialog fill



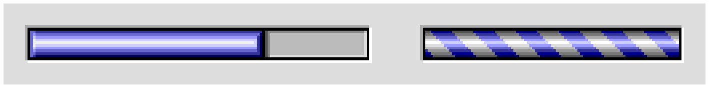
Indicator Controls

Indicator controls include large progress indicators (both determinate and indeterminate ones) and asynchronous arrows. These controls are used to convey information about application process duration and memory and disk capacity to the user.

Large Progress Indicators

Progress indicator controls, also called progress bars are used to inform the user about duration or capacity. Two types are supported: the large determinate progress indicator and the large indeterminate progress indicator. Figure 1-57 shows an example of both. Both types of progress indicators have a fixed height of fourteen pixels and a variable width.

Figure 1-57 A large determinate progress indicato and a large indeterminate one



You might use a determinate progress indicator to let the user know how much memory or disk capacity is available as you check it.

You might use a large indeterminate progress indicator, such as an animated barber pole bar, to let the user know that the application is in the process of attempting a connection or waiting to receive data, for example, during file transfer before continuing

Asynchronous Arrows

The asynchronous arrows primitive control is used to indicate that an asynchronous process is occurring in the background while the user can continue to interact with the application in the foreground. For example, the system displays them when the Find File process is searching for files. Figure 1-58 shows them in their various states.

Figure 1-58 Various states of asynchronous arrows