CHAPTER 2

Control Guidelines

This chapter provides guidelines for implementing the Mac OS Toolbox controls that a user manipulates in windows, dialog boxes, and alert boxes. Some controls give the user the ability to affect system and application activity by selecting choices and setting parameters. Other controls convey information about process duration and system capacity. Still others are used as parts of other controls or as backgrounds to them. Because these controls are provided by the Mac OS Toolbox, their appearance and behavior is standardized.

About Controls

When an operation needs additional user input before it can proceed, 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 needed input. You may also use controls in document windows to extend and enhance user interaction with your application. Wherever they appear, these controls provide users with familiar tools and formats for responding to the computer's need for information.

Standard controls include:

- Buttons (push buttons, radio buttons, bevel buttons, pop-up menu buttons, checkboxes, bevel buttons)
- Sliders and tick marks
- Clock controls and little arrows
- Disclosure triangles
- List boxes and frames
- Scroll bars
- Edit text fields and frames
- Tab controls
- Placards
- Image wells
- Progress indicators (determinate and indeterminate ones)
- Asynchronous arrows

- Group boxes (primary and secondary)
- Separator lines
- Window headers (standard and list view)
- Modeless dialog frames

Buttons

Push Buttons

A **push button** is a rounded rectangle that is labeled with text. Figure 2-1 shows some typical push buttons using platinum appearance.

Figure 2-1 Push buttons in a dialog box

Push buttons usually perform instantaneous actions, such as completing operations defined by a dialog box or acknowledging an error message. Clicking a push button (or pressing a push button's keyboard equivalent) initiates the action described by the button's name.

Always map the keyboard equivalent Command-period and the Esc (Escape) key to the Cancel push button. If you find it useful to assign keyboard equivalents to other push buttons that are used often in your application, be sure to follow the keyboard equivalent guidelines in the "Menus" chapter of *Macintosh Human Interface Guidelines*.

Push Button States

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 activated 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.

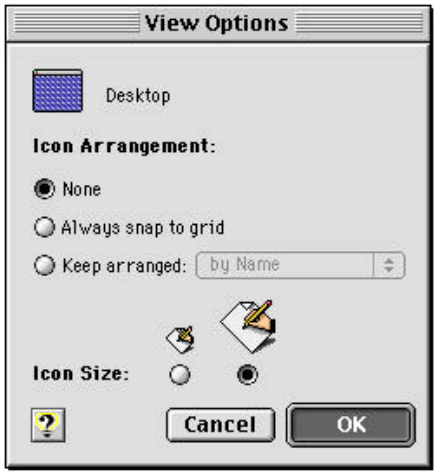
A button is displayed in its pressed state when the user clicks on it to activate its function. When the user clicks a push button, the button is highlighted (changed in appearance) to give visual feedback to the user indicating which item has been clicked.

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.

For push buttons that are activated by using a keyboard equivalent, the Dialog Manager highlights the button for eight ticks (approximately one-eighth of a second), which is long enough for the user to see that the keyboard event has taken effect. (You must highlight the Cancel button yourself 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 outside of 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 becomes highlighted again. If the user releases the mouse button while the pointer is not over the push button, nothing happens. Figure 2-2 shows a push button that is highlighted to provide feedback.

Figure 2-2 A highlighted push button



For information about implementing these behaviors for push buttons, see *Programming With the Mac OS Toolbox* and *Mac OS Toolbox Reference*.

Default Buttons

Dialog boxes and control panels which use push buttons should include a default button. The default button should be the one that the user is most likely to click. However, if the most likely choice is at all destructive (e.g. erasing a disk, deleting a file), you should define the Cancel button as the default.

The default push button displays a ring whose appearance is coordinated with the state of the button. Figure 2-3 shows the three states of standard and default push buttons.

Figure 2-3 Standard and default states of push buttons

Default push buttons are activated by pressing the Return or Enter keys. When there is no default push button, pressing Return or Enter has no effect..

Radio Buttons

Radio buttons always occur in groups. An individual radio button displays one of three states: on, off, or mixed. One button in a group is always in the on state, which is indicated by a dot in the middle of the button. To activate a radio button, the user can click the button itself or its text label. Activating a radio button turns off whichever button was previously on in that group.

Radio buttons display settings; they never initiate an action. Figure 2-4 shows a dialog box that uses radio buttons to offer icon size choices to the user.

Figure 2-4 Radio buttons for selecting the icon size

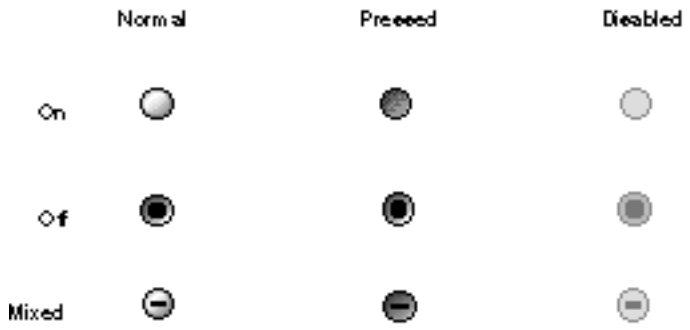
A group of radio buttons should contain from two to approximately seven items. It is possible to have more than seven buttons in a group, but you must always have at least two. Each group of radio buttons generally has a label that identifies the choices the group contains, and each button in the group usually has a name or icon that identifies its particular effect. A set of radio buttons always has the same set of choices; it is *never* dynamic (the contents do not change depending on the context).

Radio buttons represent related choices, but not necessarily opposite ones. These choices are mutually exclusive; the user can only set one button to the on state at any one time.

There is a special case called the **mixed state**, which shows that a selected range has a variety of items in the on state. For example, a set of radio buttons for selecting font size might have buttons representing 10 and 12 point sizes. If a passage of text with both 10 and 12 point text was selected, both the 10 and 12 buttons would appear in the mixed state. The ‘one button set’ rule still applies, however; if the user selected the button marked 12, all the text in the selection would change to 12 point and the mixed state would be cleared.

Radio buttons appear differently in their normal, pressed, and disabled modes depending on whether they are on, off, or in the mixed state. Figure 2-5 shows how radio buttons look in their various modes and states.

Figure 2-5 Radio button modes and states



If more than one group of radio buttons is visible at a given time, each group needs to be visually separated from the others. You should design your dialog box to allow enough space to clearly delineate multiple sets of radio buttons.

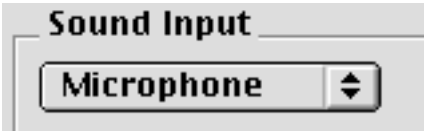
When there are several groups of controls in a dialog box, it can be useful to draw a line between a group of radio buttons and other elements. The separator line control primitive described in “Separator Lines” (page 49) is designed to do this. Group boxes (page 46) may also be used for this purpose.

For more information on laying out radio buttons in dialog boxes, see “Dialog Box Control Layout” (page 68).

Pop-Up Menu Buttons

A pop-up menu button displays a list of items which the user can select to change the state of an aspect of the application. A pop-up menu button consists of a single control. The left side of the button contains text that shows the current selection; the right side of the button shows a double triangle pointing up and down. The width of the pop-up button’s menu should be equal to or larger than the full width of the text portion of the button. Figure 2-6 shows the pop-up button in its normal state before a user selects it.

Figure 2-6 Pop-up menu button in normal state

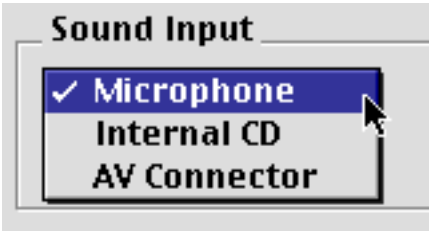


When the user clicks a pop-up menu button, a menu appears with the selected item highlighted and indicated by a checkmark. The user can move the highlighted area up and down over the menu to select a new item. Figure 2-7 shows a pop-up menu button with its list displayed.

The position of the displayed menu depends on which item has been selected. The selected item always appears at the level of the menu button, but the display of the other items depends on their relative position to the selection. In Figure 2-7, for example, the Microphone item is selected. Since it is the first item in the list, the other items appear below it. If the Internal CD item became the new selection, the display would change so that the next time the button was popped up, the Microphone item would appear above and the AV Connector item would appear below the selection.

If the user displays the pop-up menu list and releases the mouse button with the pointer outside the pop-up menu, the menu closes without changing the current selection. If the user releases the mouse button while the pointer is over a menu item, that item becomes the current selection. A checkmark appears next to the selected item and the pop-up menu button label is changed to reflect the new selection.

Figure 2-7 Pop-up menu button with displayed list.



If the user releases the mouse button after first positioning the pointer on a command and the pointer remains over the menu item for less than the user-set double-click time, the pop-up menu button becomes a sticky menu. See “Sticky Menus” (page 90) in “Menus” for information on this feature.

Checkboxes

A **checkbox** is a square with a text label next to it. The user clicks the checkbox or its label to select or deselect it. You can use one checkbox or as many as you need in a single setting.

Checkboxes are designed to provide binary choices for users. For example, a dialog box for saving files could include a checkbox labeled Compressed. In this case, the clearly implied opposite state would be an uncompressed file save.

If you want to provide a control for states which are less obviously binary, you might be better off using radio buttons, so you can use individual labels to clarify the states. It’s sometimes tempting to use a checkbox because one item takes up less space than two, but the choice may be too ambiguous for users to understand.

The default state for a checkbox is also important. In the earlier Compressed checkbox example, setting on as the default state emphasizes a preference for compressed file saves, since the user is required to take an extra action to do otherwise.

There are two options for marking the checkbox in its on state. The default option is a checkmark to indicate that a checkbox is on, but if the checkmark is not culturally acceptable, the familiar ‘X’ version of this control is available for localization purposes. When the checkbox is off, it is unmarked.

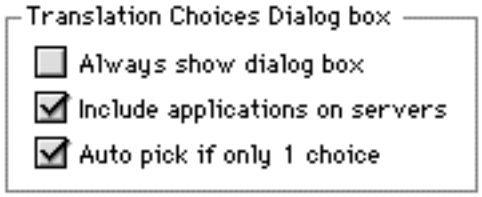
There is a mixed state for checkboxes, which shows that a selected range of items has some in the on state and some in the off state. For example, a text formatting checkbox for bold text would be in the mixed state if a text selection contained both bold and non-bold text.

Figure 2-8 shows how checkboxes look in their various modes and states.

Figure 2-8 Checkbox modes and states

Checkboxes differ from radio buttons in that they are independent of each other, even when they offer related options. Any number of checkboxes can be on, off, or mixed at the same time. Figure 2-9 includes a group of checkboxes with two options selected concurrently.

Figure 2-9 A set of checkboxes with concurrent selections



It is important to arrange checkboxes in clearly organized groups. Allow ample white space between checkbox groups and other controls. Use separator lines and group boxes when appropriate. For more information on laying out controls, see “Dialog Box Control Layout” (page 68)..

Bevel Buttons

A bevel button is a rectangular control with a beveled edge which gives the button a three-dimensional appearance. You can choose from among three sizes of bevels—small, medium, and large—for any size button you create. The small bevel is two pixels wide; the medium bevel is three pixels wide; and the large bevel is four pixels wide. Figure 2-10 shows these bevel sizes.

Figure 2-10 Bevel buttons with small, medium, and large bevels

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 aligned according to the system default script direction; if the system software displays text from right to left, the button text will be right-aligned. Bevel buttons containing both text and an image will place the image correctly according to the system default script direction if the image and text are on the same horizontal plane.

Bevel buttons can exist in seven states. There are five active states and two disabled. The active states are:

- Off
- Pressed (Was Off)
- On
- Pressed (Was On)
- Mixed, for use when behaving as a checkbox or radio button.

Two pressed states are provided for those cases in which it is necessary for an application to determine the previous state of a bevel button which is now pressed.

Disabled bevel buttons can be shown as off or on.

These states are depicted in Figure 2-11.

Figure 2-11 Bevel button states

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 pop-up button. 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.

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. You can use any number of bevel buttons as push buttons in your dialog box or window, just as you can use any number of standard push buttons together. Figure 2-12 shows two bevel buttons used in this way.

Figure 2-12 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 when released. If the user rolls the cursor off the button at any time while holding down the mouse button, the bevel button reverts to its off state.

For more information on push button behavior, see “Push Buttons” (page 20).

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. Bevel buttons acting as radio buttons are used in groups, just as standard radio buttons are. As with a group of standard radio buttons, only one button may be set by the user at any given time. The behavior for standard radio buttons, described in “Radio Buttons” (page 23), applies to radio bevel buttons. Figure 2-13 shows a group of bevel buttons serving as radio buttons in a text justification toolbar.

Figure 2-13 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 checkbox behavior, described in “Checkboxes” (page 27), while retaining the bevel button appearance. The on, off and mixed states are available when you use bevel buttons as checkboxes. You can use checkbox bevel buttons singly or in sets.

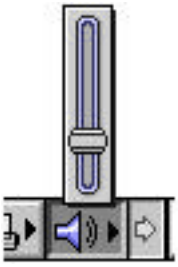
Figure 2-14 shows a group of bevel buttons used as checkboxes in a text styling tool. The buttons labeled B for boldface and U for underline in the on state because the selected range of text contains both boldfaced and underline characters.

Figure 2-14 A set of bevel buttons used as checkboxes

Bevel Buttons as Pop-up Buttons

A bevel button can be used as a pop-up button with or without a menu. Figure 2-15 shows a pop-up bevel button that serves as a standard pop-up button with a slider instead of a menu.

Figure 2-15 A pop-up bevel button used with a slider



When a bevel button behaves as a pop-up menu betton, the user clicks on the bevel button to display a pop-up menu directly below or to the right of the button. If used as a standard menu, the menu closes immediately after the user releases the mouse button. If you enable sticky menu behavior, the menu remains displayed after the user clicks on the button. The section on “Pop-Up Menu Buttons” (page 25) describes this more fully. Figure 2-16 shows an example of a pop-up bevel button with a sticky menu.

Figure 2-16 A pop-up bevel button with sticky menu

You can also set up a bevel button which the user clicks once to simply turn on the function represented by the button, but if the bevel button is clicked and held, a pop-up menu is displayed which offers further options for that function. This behavior is useful for providing option sets in tool palettes.

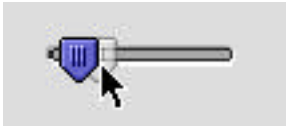
You can specify that a pop-up bevel button displays either a small or large arrow to indicate its pop-up behavior. These arrows can be either horizontally or vertically oriented. Figure 2-15 shows pop-up bevel buttons with large horizontally-oriented arrows. Figure 2-16 shows pop-up bevel buttons with small vertically-oriented arrows.

Sliders and Tick Marks

A **slider** control consists of a slider bar, which displays a range of allowable values, and an indicator, which marks the current setting and allows the user to define a new value.

Sliders can be horizontal or vertical. The indicator can point in any direction or be nondirectional. The user drags the indicator 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 is a copy of the indicator which shows where the user has dragged the indicator. Figure 2-17 shows a large slider with a downward pointing indicator shadowed by a ghost indicator.

Figure 2-17 Large slider and ghost indicator



Built into the slider control are tick marks that you can use to represent incremental values. Tick marks are optional; you can specify whether or not they should be drawn with the slider. You determine thenumber of tick marks and what fraction of the scale each mark represents. If you use tick marks, they are drawn appropriately for the direction of the slider. Figure 2-18 shows a horizontal slider using vertical tick marks.

Figure 2-18 A horizontal slider with vertical tick marks

Be sure to label the tick marks so that they clearly indicate the directions in which the indicator move and what effect this has on the setting. Plain tick marks signify the directions in which the indicator moves, but they do not necessarily show in which direction the value increments. While it is true that most people assume that moving an indicator up a vertical slider means increasing the value of the setting, you can easily remove all doubt with graphics or text. Figure 2-19 shows an example of a horizontal slider that uses incremental numbering to indicate increasing values as the user moves the indicator to the right.

Figure 2-19 A slider with direction information

Sliders support live feedback, a process also known as “live dragging.” This allows you to design a dialog in which the user is given constant visual updates of the changes in value as the indicator is moved, as opposed to the standard behavior of waiting to update the value until the indicator is released.

Make sure that you don’t use a scroll bar when you really should 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, while the scroll indicator represents the relative position of the window over the length of the document. Using a scroll bar to change a setting confuses the meaning of the element and makes the interface inconsistent. For more information on using scroll bars, see the “Windows” chapter of *Macintosh Human Interface Guidelines* for details.

Little Arrows

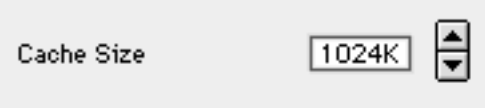
The control consisting of two arrows pointing in opposite directions is commonly called **little arrows.** Little arrows provide the user with a means of increasing or decreasing values in a series. Figure 2-20 shows the little arrows control in its normal state, with the up arrow depressed, with the down arrow depressed, and in its disabled state.

Figure 2-20 Little arrows in various states

The little arrows control has a label that specifies the content to which it relates. If the user single-clicks the up arrow, the value displayed is incremented by one unit of change. If the user clicks and holds the arrow, the value increases or decreases until the user releases the mouse button. While the user clicks the arrow, it is highlighted to provide feedback to the user.

The unit of change depends on the content. If the content area displays years, the increment is one year. If you used little arrows to control the size of a RAM cache, however, you might use multiples of 32K as the increment (as in Figure 2-21.)

Figure 2-21 Little arrows used to control RAM cache



Clock Controls

The clock control combines the edit text field and little arrows control into a convenient implementation of the familiar Date & Time control panel. The user can change the time and date displayed by using the little arrows or by typing the value into the edit text field. Figure 2-22 shows a clock control displaying a date.

Figure 2-22 Clock control displaying date

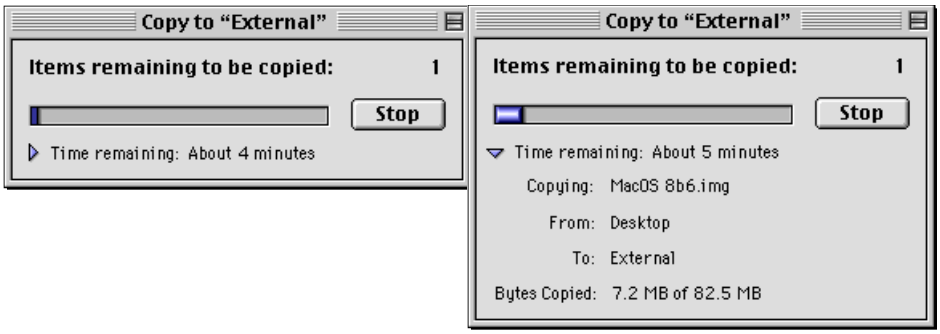
You can make a clock control inactive, so that it displays time and date values without allowing the user to change them.

Disclosure Triangles

The **disclosure triangle** is a control that allows the display or "disclosure" of information which elaborates on the primary information in a window.

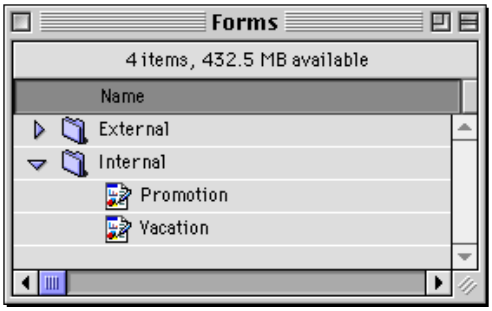
One way to use disclosure triangles is to provide a way for users to expand a dialog box or control panel. When the user clicks on the disclosure triangle, the triangle rotates downward and the window expands to provide supplemental information. Clicking on the triangle again rotates the triangle back to the right and restores the original appearance of the window. Figure 2-23 shows a file copy dialog box in its default configuration and with the disclosure triangle turned to reveal additional details of the copy operation.

Figure 2-23 A disclosure triangle revealing additional information



Another familiar implemention of disclosure triangles is in the Finder's list view. The triangle appears next to the icon of each folder in the window. The user clicks the triangle to expand the view by revealing a list of the contents of the folder without actually opening it. The triangle rotates downward when the folder is expanded, which tells the user that the view is expanded even in cases when the folder is empty. Clicking the triangle again restores the view to its original (collapsed) state and turns the triangle back to the right. Figure 2-24 shows the disclosure triangle in expanded and collapsed positions.

Figure 2-24 Disclosure triangles used in Finder list view



This use of a disclosure triangle is also known as an “outline triangle”, since it provides a function similar to the Outline view used in many applications.

In the Finder’s list view, a disclosure triangle may be turned to the open position by using the Command-right arrow keyboard equivalent and closed by using Command-left arrow. If you use disclosure triangles in your application to provide expanded / collapsed list views, you should also provide the keyboard equivalents.

List Boxes and Frames

The list box control is a complete solution for creating scrolling lists. It provides one or two embedded scroll bars and a scrolling list box which features a white background and a two-pixel-wide rectangular frame whose inside lines share the outside lines of the scroll bar(s). Figure 2-25 shows a list box in use.

Figure 2-25 A list box

If you need to provide extra features in a scrolling list, such as icons, the list box frame is available as a separate control so that non-standard list boxes can be made Appearance-compliant.

When you create the list of items in a scrolling list, you may find item names that are too long to fit in the list. When this is the case, it's best to eliminate characters in the middle of the item's name and insert ellipsis points there, preserving the beginning and end of the name. This technique is known as "center truncation." Users often add version numbers to the end of their document or device names, so if you cut off the end of the item name instead of using center truncation, the version number is lost and the user must guess which of several similarly named items 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 for contiguous sets of items or Command-clicking for non-contiguous sets) 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.

Scrolling lists can accept keyboard input for navigating within the list. The user can use the arrow keys to move through the list one item at a time in the direction of the arrow. Users can also select an item from the list by typing the beginning character or characters of its name; this technique is called **type selection.**

Scrolling lists are not appropriate to provide choices in a limited range. Because the full range may not be 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 letting users choose their preference in the range. See "Sliders and Tick Marks" (page 33) for information about sliders.

Scroll Bars

Scroll bars allow users to view areas of a document or a list that is larger than can fit into the current window. A scroll bar is a shaded gray rectangle that has a black arrow in a box at each end. Inside the gray area is an indicator (also know as a scroll box) which indicates the relative position of the currently visible portion of the content being scrolled.

You can use scroll bars with windows as well as with scrolling lists. For more information on the use of scroll bars with windows, see “Windows” in *Macintosh Human Interface Guidelines*.

Figure 2-26 shows an example of a horizontal scroll bar. Note that the scroll indicator takes the accent color defined in the Appearance control panel.

Figure 2-26 A horizontal scroll bar

Edit Text Fields

The **edit text field** (also known as a text entry field) is a rectangular area in which the user enters text or modifies existing text. The edit text field can be disabled and enabled. It allows keyboard filtering and supports password entry.

Figure 2-27 shows an edit text field with a label.

Figure 2-27 An edit text field with label

It is up to you to provide appropriate editing services for a edit text field. For more information about this, see “Text Entry Fields” in *Macintosh Human Interface Guidelines*.

The edit text frame is used to surround an edit text field. It is useful for making a non-standard version of the edit text field Appearance-compliant. The edit text frame is two pixels wide. It has two states, enabled and disabled. Figure 2-28 shows an example of an edit text frame.

Figure 2-28 A edit text frame

Static Text Fields

A static text field embeds static, unchangeable text in dialog boxes. Static text fields may be made active or inactive. Figure 2-29 shows a static text field in both states.

Figure 2-29 A static text field in active and inactive states

Tab Controls

The tab control provides a convenient way to present information in a multi-page format. This control is distinguished by a group of buttons that give the visual appearance of folder tabs. The user selects the desired page by clicking the appropriate tab.

The tab control is available in two sizes. It currently supports one row of tabs running along the top, as shown in Figure 2-30 and Figure 2-31. You specify the names and icons that label the tabs. Figure 2-30 shows the tab control with 12-point font labels.

Figure 2-30 Large tab control with 12-point font labels

Figure 2-31 shows the small tab control with 10-point font tab labels.

Figure 2-31 Small tab control with 10-point font tab labels

Clicking an inactive tab changes the tab to the active state. Releasing the mouse button with the cursor over the tab will display the page for the now-active tab. Dragging the cursor outside the clickable region and releasing the mouse button will make the tab revert to the inactive state without changing the page displayed. Figure 2-32 shows the active state of a tab.

The appearance of the content area of a tab control (also known as a pane) depends on its implementation. Figure 2-32 shows a tab control inside a control panel. In this implementation, the sides of the pane appear to be ‘tucked’ under the edge of the content region by one pixel.

Figure 2-32 Tab control with sides tucked under edge of content region

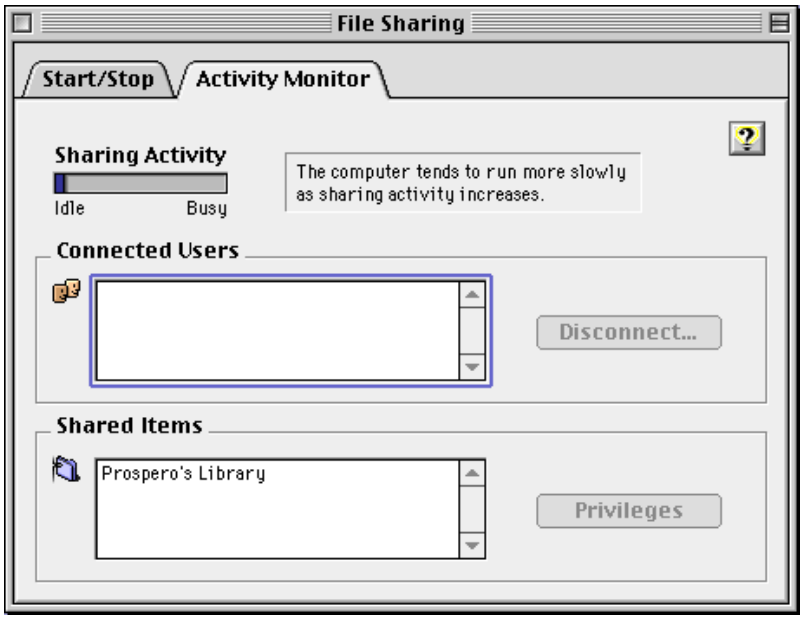


Figure 2-33 shows a tab control used in a modal dialog box. Note that the left and right sides of the pane are inset two pixels from the edge of the dialog box’s content region. This small distinction helps emphasize the fact that the tab is part of a dialog box. Contrast this with Figure 2-34, which shows a tab control with tucked edges and a scrollable content area.

Figure 2-33 Tab control used in a modal dialog box

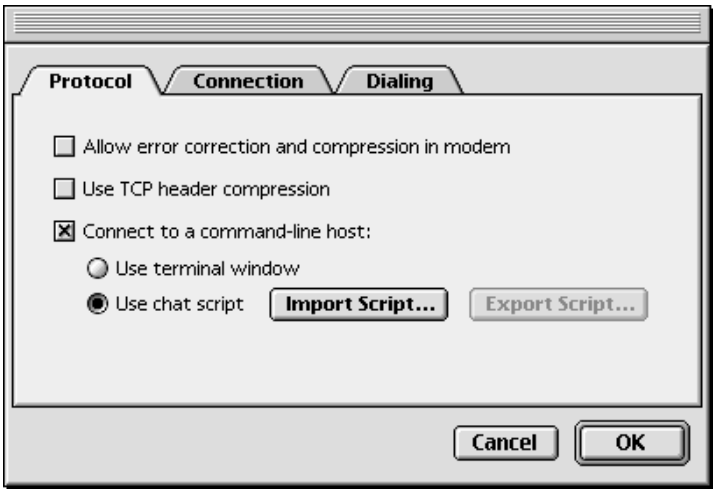
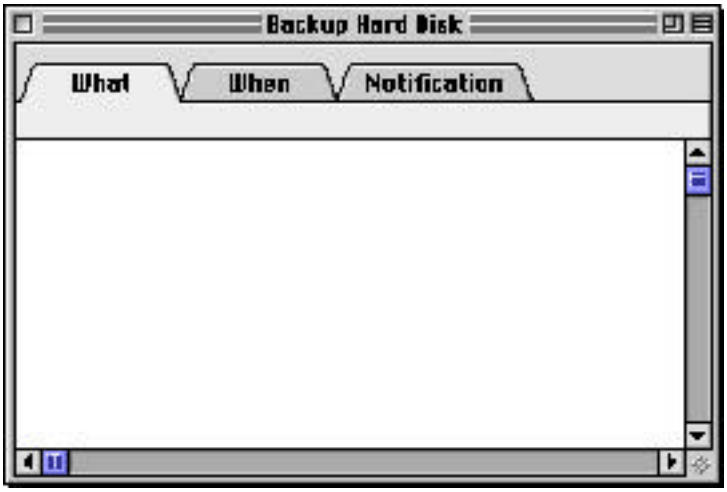


Figure 2-34 Tab control with tucked edges and a scrollable content area



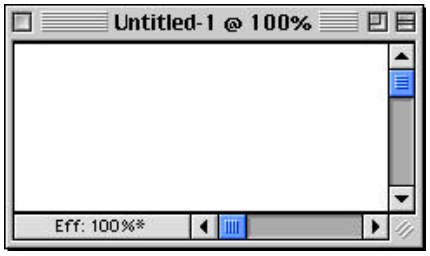
Controls such as push buttons or scroll bars may be embedded into tab controls. These embedded controls may be **global**, which means they are available to and affect the settings of all the panes in a set of tabs. Controls may also be embedded within an individual pane, in which case they affect only the settings displayed in that pane. It is important that you make this distinction unambiguous to the user through clear, specific labeling and placement of controls. In Figure 2-33, for example, the push buttons at the bottom are clearly global, while the checkboxes affect only the active pane.

Placards

A placard is a control that you can use as an information display or as background fill for a user control area. Placards have three states: normal, pressed and disabled. The disabled state is appropriate for the background fill function; it indicates to the user that the area does not allow interaction.

Perhaps the most familiar use of the placard control is as a small information panel, often placed at the bottom of a window to the left of the horizontal scroll bar. Figure 2-35 shows an example of this use of a placard.

Figure 2-35 A placard used to report information to the user



You can extend the functionality of a placard. Providing a pop-up menu, for instance, would give the user a convenient way to choose which type of information to display on the placard.

Image Wells

The image well control is used to display non-text visual content, such as icons or pictures. The image well frame is two pixels wide. This control has three states: enabled, disabled and selected. Figure 2-36 shows the enabled and selected states.

Figure 2-36 The image well in its enabled and selected states

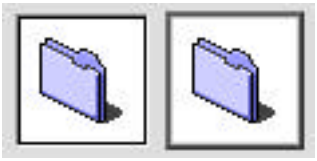


Image wells can serve as drag and drop targets. You could use a set of image wells to manage thumbnail images in a clip-art catalog, for instance, or display a set of user-selectable icons.

Group Boxes

Group boxes are used to associate, isolate and distinguish groups of related items in a dialog box. You can embed other controls, such as radio buttons, checkboxes, and popup menu buttons, within group boxes.

Group boxes are defined as either primary or secondary. The main visual distinction between the two types is their borders; primary group box border lines are two pixels wide with an etched look, while secondary group box borders are one pixel wide.

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 pop-up menu title, or a check box title. Figure 2-37 shows an untitled primary group box.

Figure 2-37 An untitled primary group box

Figure 2-38 shows a titled primary group box with a scrollable list.

Figure 2-38 A titled primary group box

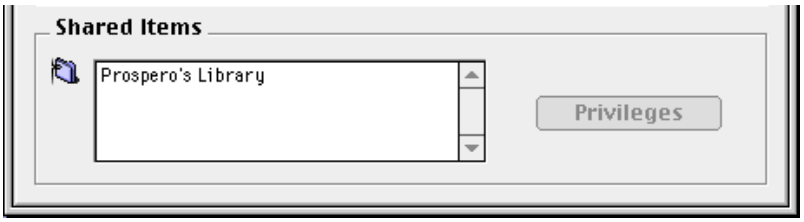


Figure 2-39 shows a primary group box using a pop-up menu title. This is useful for displaying a variety of related settings in a limited space.

Figure 2-39 A primary group box with a pop-up menu title

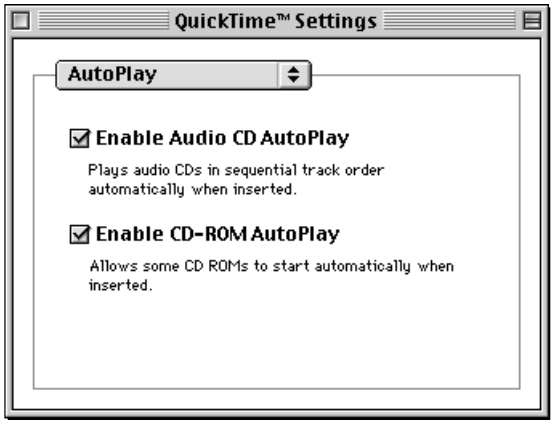
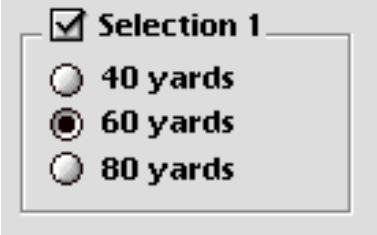


Figure 2-40 shows a primary group box with a checkbox title. This is useful to indicate when a group of settings may be disabled by the user.

Figure 2-40 A primary group box with a checkbox title



Secondary group boxes almost always appear inside of primary ones. Secondary group boxes are generally used for nesting and grouping together subsidiary information. Do not use secondary group boxes in place of primary ones; this produces inconsistent dialog box appearances which confuse users.

Figure 2-41 A secondary group box nested inside a primary one

Separator Lines

The separator line is used in dialog boxes to delineate horizontal or vertical regions of the content area. It provides an Appearance-compliant alternative to drawing a QuickDraw line.

The 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 2-42 shows a horizontal separator line.

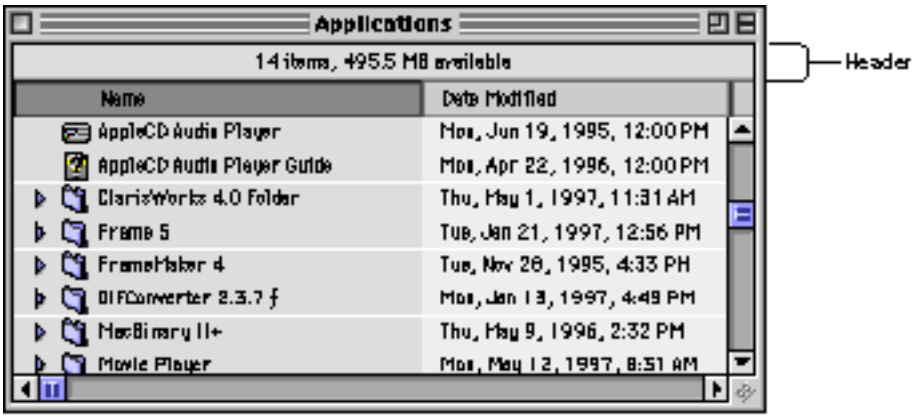
Figure 2-42 A horizontal separator line

Window Headers

The window header is designed to be a simple way to present information about a window's contents. It is a familiar feature of the Finder, but it can also be implemented in document windows.

The 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. There is a list view variant which removes the bevel. Figure 2-43 shows an example of a header, placed just below the title bar, and the list view variant just below it.

Figure 2-43 A Finder window using headers



Modeless Dialog Frames

The modeless dialog frame is used in the content region of a modeless dialog box. It is a rectangle with a two-pixel-wide border which shares the inside lines of the document window. This control provides a beveled appearance which

helps distinguish the modeless dialog box from other windows. The modeless dialog frame has foreground and background states.

Figure 2-44 shows a modeless dialog frame in a dialog box.

Figure 2-44 A modeless dialog frame

Progress Indicators

Progress indicators, also called progress bars, are used to inform the user about duration or capacity. Two types are supported: the determinate progress indicator and the indeterminate progress indicator. Figure 2-45 shows an example of each indicator. Both types have a fixed height of fourteen pixels and a variable width.

Figure 2-45 Progress indicators

The determinate progress indicator should be used in cases when the full scope of an operation can be determined. The bar moves from left to right as the operation progresses, so the user can easily see how much of the process has been completed. You might use a determinate progress indicator to show the progress of a file copying operation.

The indeterminate progress indicator is intended for those cases where the duration of a process is not easily determined. This indicator consists of a striped cylinder that spins to indicate an ongoing process. It provides less information to the user and therefore should be reserved for cases when the determinate indicator is clearly inappropriate. You might use an indeterminate progress indicator to let the user know that the application is in the process of attempting a communication connection or is waiting to receive data during file transfer.

For more information on the behavior of progress indicators, see *Macintosh Human Interface Guidelines.*

Asynchronous Arrows

The asynchronous arrows control (also known as chasing arrows) is a simple animation used to indicate that an asynchronous background process is occurring; in other words, a process which does not display a dialog box that might contain a progress indicator. For example, the Find File utility displays asynchronous arrows as a cursor while it is searching for files. Figure 2-46 shows asynchronous arrows in their various states.

Figure 2-46 Various states of asynchronous arrows