Menu Manager

Contents

Contents

You can use the Menu Manager to create and manage the menus in your application. Menus allow the user to view or choose from a list of choices and commands that your application provides.

All Macintosh applications should provide these standard menus: the Apple menu, the File menu, and the Edit menu. If you include an Apple menu as a menu of your application, the Menu Manager automatically adds the Help and Application menus to your application's menu bar; it adds the Keyboard menu if more than one keyboard layout or input method is installed.

Menus are typically stored as resources. This chapter describes the menu-related resources. See the chapter "Introduction to the Macintosh Toolbox" in this book for general information on resources and see the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox* for information on Resource Manager routines. See *Macintosh Human Interface Guidelines* for additional examples of menus that incorporate many principles of user interface design. *Inside Macintosh: Text* contains further information on localizing your application for worldwide markets.

You can choose to provide help balloons for your application's menus. See the chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox* for additional details on providing help balloons for your application's menus.

You often present a dialog box to the user as a result of the user's choice of a menu command that requires additional information before you can perform the command. See the chapter "Dialog Manager" later in this book for information on creating dialog boxes in your application.

For additional information on processing events, see the chapter "Event Manager" earlier in this book.

This chapter provides an introduction to menus and the menu bar, and it then describes

- various types of menus your application can use
- standard menus
- how to store menus as resources
- how to create menus
- how to create a menu bar
- how to change characteristics of menu items
- how to add items to a menu

Introduction to Menus

A **menu** is a user interface element you can use in your application to allow the user to view or choose an item from a list of choices and commands that your application provides. Menus can appear in several different forms: pull-down menus, hierarchical menus, and pop-up menus.

A **pull-down menu** is identified by a menu title (a word or an icon) in the menu bar. Your application can use pull-down menus in the menu bar to allow users to choose a command or perform an action on a selected object. A **pop-up menu** is a menu that does not appear in the menu bar, but appears elsewhere on the screen when the user presses the mouse button while the cursor is in a particular place. Pop-up menus are most often accessed from a dialog box. Your application can use pop-up menus to let the user select one choice from a list of many or to set a specific value. A **submenu** refers to a menu that is attached to another menu. A menu to which a submenu is attached is referred to as a **hierarchical menu.**

Figure 3-1 shows examples of a pull-down menu, a submenu, and a pop-up menu.

Figure 3-1   A pull-down menu, a submenu, and a pop-up menu
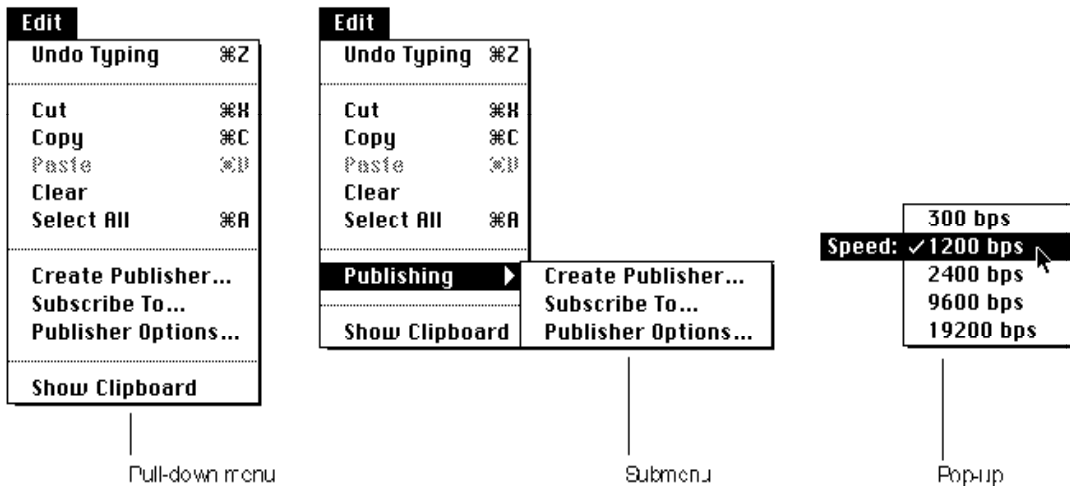
The standard **menu bar** extends across the top of the startup screen and contains the title of each available pull-down menu. Your application's menu bar should always provide at least the Apple menu, the File menu, and the Edit menu. When you insert the Apple menu in your application's menu bar, the Menu Manager automatically adds the Help and Application menus to your application's menu bar. It also adds the Keyboard menu if multiple script systems are installed or if a certain bit is set in the `'itlc'` resource. Your application can include as many other menus as fit on the smallest screen on which your application runs, and you should create only as many items as are essential to your application.

If your application uses a menu bar, you should make it always visible and available for use. If you do not always wish to display the menu bar (for example, if your application allows the user to view a screen presentation), you can give the user the option of viewing the presentation on the entire screen without the menu bar showing. However, you must provide a way, such as a keyboard equivalent for a command, for the user to access the menu bar or to make the menu bar reappear.

Using menus in your application allows the user to explore many possible choices and options without having to choose any particular one. By providing help balloons for

Menu Manager

your menus, you further allow users to learn about the possible actions or consequences of a particular menu choice without having to choose the menu command to find out what happens.

Figure 3-2 shows the SurfWriter application's menu bar with the Edit menu displayed. This application supports the standard Apple, File, and Edit menus; the Help and Application menus; and in addition supports two other application-specific menus.

Figure 3-2   The SurfWriter application's menu bar with the Edit menu displayed
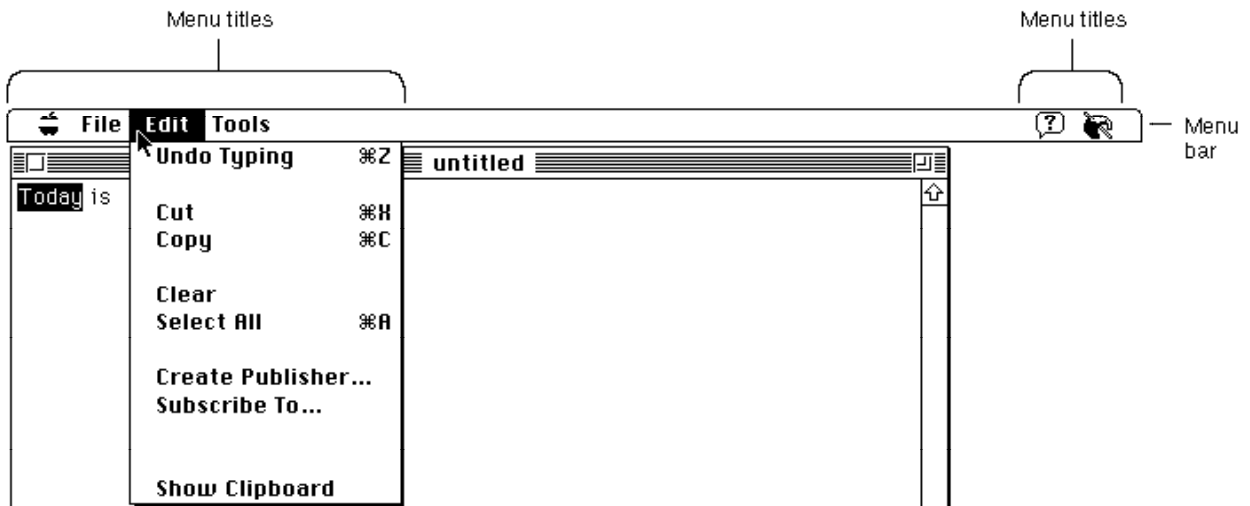
Each menu has a menu title and one or more menu items associated with it. You should name each menu so that the title describes or relates to the actions the user can perform from that menu. For example, the Edit menu of a typical application contains commands that let the user edit the contents of a document.

Your application can disable any menu. The Menu Manager indicates that a menu is disabled by dimming its menu title. (In Figure 3-2, the Colors menu is disabled.) The Menu Manager dims all menu items of a disabled menu. The user can still pull down and examine the items in a disabled menu, but cannot choose any of the items.

Your application can also disable individual menu items. The Menu Manager dims the appearance of a disabled item and does not highlight it when the user rests the cursor on that item. If the user releases the mouse button while the cursor is over a disabled menu item, the Menu Manager reports that the user did not choose a menu command. (You can determine if this happened, however, by using the MenuChoice function.)

In Figure 3-2, the Paste command is disabled; the SurfWriter application disables the Paste command if the Clipboard is empty. SurfWriter also disables the Publisher Options command when the current selection does not contain a publisher or a subscriber. As explained in the chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox*, your application should provide help balloons for disabled items that describe what the item normally does and explain why the item is not available at this time.

Note

Although `enabled` and `disabled` are the constants you use in a
resource file to display or to dim menus and menu items, you shouldn't
use these terms in your help balloons or user documentation. Instead
use the terms *menus*, *menu commands*, and *menu items* for those that are
enabled, and use the terms *not available* and *dimmed* to distinguish those
that have been disabled. ◆

The Menu Manager highlights an enabled menu item when the cursor is over it.
Enabled items do not have a dim appearance and can be chosen by the user.

Your application specifies whether menu items are enabled or disabled when it first
defines and creates a menu. You can also disable or enable menu items at any time after
creating a menu. You should enable a menu item whenever your application allows the
user to choose the action associated with that item, and you should disable an item
whenever the user cannot choose that item. For example, if the user selects text and then
presses the mouse button while the cursor is in the menu bar, you should enable the
Copy command in the Edit menu. You should disable the Copy command in the Edit
menu if the user has not selected anything to copy.

Your application can also specify other characteristics of menu items, such as whether
the item has a marking character next to its text (for example, a checkmark) or whether
the item has a keyboard equivalent (for example, Command-C for the Copy command).
"Menu Items" beginning on page 3-12 describes the characteristics of individual menu
items in more detail.

The user typically chooses commands by moving the cursor to the menu bar and
pressing the mouse button while the cursor is over a menu title. When the user presses
the mouse button while the cursor is in the menu bar, your application should call the
`MenuSelect` function. The `MenuSelect` function tracks the mouse, displays and
removes menus as the user drags the cursor through the menu bar, highlights menu
titles as the user drags the cursor over them, displays the menu items associated with a
selected menu, highlights enabled menu items as the user drags through a menu, and
handles all user activity until the user releases the mouse button.

The user chooses a menu item by releasing the mouse button while the cursor is over a
particular enabled menu item. When the user chooses a menu item, the Menu Manager
briefly blinks the chosen menu item (to confirm the choice) and then removes the menu
from the display. The Menu Manager leaves the title of the chosen menu highlighted to
provide feedback to the user.

The `MenuSelect` function returns information that allows your application to
determine which menu item was chosen. Your application then typically responds by
performing the desired command. When your application completes the requested
action, your application should unhighlight the menu title, indicating to the user that the
action is complete.

The user can move the cursor out of the menu (or menu bar) at any time; the Menu
Manager displays any currently visible menu as long as the mouse button is pressed. (If
the cursor is outside of the menu, the Menu Manager removes any highlighting of the
menu item.) If the user releases the mouse button outside of a menu, the `MenuSelect`

function reports that the user did not choose a menu item, and the Menu Manager removes any currently visible menu. Your application should not take any action if the user does not choose a menu item.

Menu and Menu Bar Definition Routines

The menu definition procedure and menu bar definition function define the general appearance and behavior of menus. The Menu Manager uses these routines to display and perform basic operations on menus and the menu bar.

A **menu definition procedure** performs all the drawing of menu items within a menu. When you define a menu, you specify its menu definition procedure. The Menu Manager uses the specified menu definition procedure to draw the menu items in a menu, determine which item the user chose from a menu, insert scrolling indicators as items in a menu, and calculate the menu's dimensions.

A **menu bar definition function** draws the menu bar and performs most of the drawing activities related to the display of menus when the user moves the cursor between them. Unless you specify otherwise, the Menu Manager uses the standard menu bar definition function to manage your application's menu bar. The Menu Manager uses the standard menu bar definition function to draw the menu bar, clear the menu bar, determine whether the cursor is in the menu bar or any currently displayed menu, calculate the left edges of menu titles, highlight a menu title, invert the entire menu bar, erase the background color of a menu and draw the menu's structure (shadow), and save or restore the bits behind a menu.

Apple provides a standard menu definition procedure and standard menu bar definition function. These definition routines are stored as resources in the System file. The standard menu definition procedure is the `'MDEF'` resource with resource ID 0. The standard menu bar definition function is the `'MBDF'` resource with resource ID 0.

When you define your menus and menu bar, you specify the definition routines that the Menu Manager should use when managing them. You'll usually want to use the standard definition routines for your application. However, if you need a feature not provided by the standard menu definition procedure (for example, if you want to include more graphics in your menus), you can choose to write your own menu definition procedure. See "Writing Your Own Menu Definition Procedure" beginning on page 3-87 for more information. While the Menu Manager does allow you to specify your own menu bar definition function, Apple recommends that you use the standard menu bar definition function.

The Menu Bar

Each application has its own menu bar. The menu bar of an application applies to only that application. You usually define a menu bar for your application by providing a **menu bar** (`'MBAR'`) **resource** that lists the order and resource ID of each menu that appears in your menu bar. You define the menu title and the individual characteristics of menu items that appear in a menu by providing a **menu** (`'MENU'`) **resource** for each

menu that appears in your menu bar. You use Menu Manager routines to create the menus and menu bar based on these resource definitions.

Your application can change the enabled state of a menu, add menus to or remove menus from its menu bar, or change the characteristics of any menu items. Whenever your application changes the enabled state of a menu or the number of menus in its menu bar, your application must call the `DrawMenuBar` procedure to update the menu bar's appearance.

The menu bar (as defined by the standard menu bar definition function) is white, with a height that is tall enough to display menu titles in the height of the system font and system font size, and with a black lower border that is one pixel tall. The menu bar is as wide as the screen and always appears on the monitor designated by the user as the startup screen. (The user selects a startup screen using the Monitors control panel.) The menu bar appears at the top of the screen, and nothing except the cursor can appear in front of it. Figure 3-3 shows the menu bar of the SurfWriter application.

Figure 3-3   The menu bar of the SurfWriter application
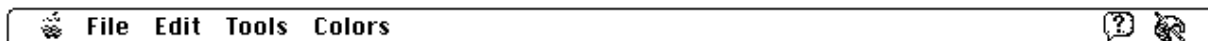
The menu bar helps to indicate the active application. The active application is the one whose menu bar is currently showing and whose icon appears as the menu title of the Application menu.

The titles of menus appear in the menu bar. A menu title is a text string (except for the Apple, Help, Keyboard, and Application menus, the titles of which contain a small icon). Menu titles always appear in the system font and system font size (for Roman scripts, the system font is Chicago and the system font size is 12).

You can insert any number of menu titles in the menu bar; however, less than 10 is usually optimum. Keep in mind that not all users have the same size monitor. Design your menu bar so that all titles can fit in the menu bar of the smallest screen on which your application can run. You should also consider localization issues when designing the number of menus that fit in your menu bar—not all menu titles might fit in the menu bar once the menu titles are translated. For example, English text often grows 50 percent larger when translated to other languages.

Figure 3-4 shows the SurfWriter application's menu bar with menu titles that have been localized for another script system.

Figure 3-4   The SurfWriter application's menu bar localized for another script system

Menus

A menu (as defined by the standard menu definition procedure) is a list of menu items arranged vertically and contained in a rectangle. The rectangle is shaded and can extend vertically for the length of the screen. If a menu has more items than will fit on the screen, the standard menu definition procedure adds a downward-pointing triangular indicator to the last item on the screen, and it automatically scrolls through the additional items when the user moves the cursor past the last menu item currently showing on the screen. When the user begins to scroll through the menu, the standard menu definition procedure adds an upward-pointing triangular indicator to the top item on the screen to indicate that the user can scroll the menu back to its original position.

Each menu can have color information associated with it. If you do not define the colors for your menus in your application's menu color information table, the Menu Manager uses the default colors for your menus and menu bar. The default colors are black text on a white background. In most cases the default colors should meet the needs of your application. "The Menu Color Information Table Record" on page 3-98 and "The Menu Color Information Table Resource" on page 3-155 give information on how you can define colors for your application's menus.

Your application's menus can contain any number of menu items. "Menu Items" (the next section) describes the visual variations that you can use when defining your menu items.

You typically define the order and resource IDs of the menus in your application's menu bar in an `'MBAR'` resource. You should define your `'MBAR'` resource such that the Apple menu is the first menu in the menu bar. You should define the next two menus as the File and Edit menus, followed by any other menus that your application uses. You do not need to define the Keyboard, Help, or Application menus in your `'MBAR'` resource; the Menu Manager automatically adds them to your application's menu bar if your application calls the `GetNewMBar` function and your menu bar includes an Apple menu or if your application inserts the Apple menu into the current menu list using the `InsertMenu` procedure.

You define the menu title and characteristics of each individual menu item in a `'MENU'` resource. "Creating a Menu Resource" on page 3-43 describes the `'MENU'` resource in more detail.

Pop-up menus do not appear in the menu bar but appear elsewhere on the screen. You often use pop-up menus in a dialog box when you want the user to be able to make a selection from a large list of choices. For example, rather than displaying the choices as a number of radio buttons, you can use a pop-up menu to display the choices at the user's convenience.

A hierarchical menu refers to either a pull-down or pop-up menu that has a submenu attached to it. (However, you should avoid attaching a submenu to a pop-up menu whenever possible, as this can make the interface more complex and less intuitive to the user.)

"Creating a Pop-Up Menu" on page 3-56 gives additional information about pop-up menus, and "Creating a Hierarchical Menu" on page 3-53 describes hierarchical menus in more detail.

Menu Items

A **menu item** can contain text or can be a line (a **divider**) separating groups of choices. A divider is always dimmed, and it has no other characteristics associated with it.

Each menu item (other than dividers) can have a number of visual characteristics:

■ An icon to the left of the menu item's text. If you define an icon for a menu item, use an icon that gives a symbolic representation of the menu item's meaning or effect. You can specify an icon, a small icon, a reduced icon, or a color icon as the icon for a menu item; however, items with small or reduced icons cannot have submenus and cannot be drawn in a script other than the current system script.

■ A checkmark or other marking character to the left of the menu item's text (and to the left of the item's icon, if any). Use such a mark if you need to denote the status of the menu item or the mode it controls. A menu item can have a mark or a submenu, but not both.

■ The symbol for the Command key (⌘) and another 1-byte character to the right of the menu item's text (referred to as the *keyboard equivalent* of a command). Use this if your application allows the user to invoke the menu command from the keyboard by pressing the Command key and one or more other keys in combination, just as if the user had chosen the command from the menu. An item that has a keyboard equivalent cannot have a submenu, a small icon, or a reduced icon and cannot be drawn in a script other than the current system script.

■ A triangular indicator to the right of the menu item's text to indicate that the item has a submenu. A menu item that has a submenu cannot have a keyboard equivalent, a marking character, a small icon, or a reduced icon and cannot be drawn in a script other than the current system script.

■ A font style—either plain or one of various other styles—for the menu item's text. You can set the menu item's style to bold, italic, underline, outline, shadow, or any combination of these.

■ The text of the menu item. Choose words for menu items that declare the action that occurs when the user chooses the command (usually verbs, such as Print or Save). You can also use adjectives if the command changes the attribute of a selected object (for example, Bold or Italic). Unless you specify otherwise, the text of menu items appears in the script of the system font and system font size (for Roman scripts, the system font is Chicago and the system font size is 12 points). If you want a menu item's text to appear in a script other than the current system script, you can specify a script code for the text. The Menu Manager draws the item's text in the script identified by the script code if the script for the specified script system is installed. A menu item that is drawn in another script cannot have a submenu, small icon, or reduced icon.

■ Three ellipsis points (…) as the last character in the text of the menu item. Use ellipses in the text of menu items to indicate that your application displays a dialog box that requests more information from the user before executing the command. Do not use

ellipses in menu items that display informational dialog boxes that do not require additional information from the user. In addition, you should not use ellipses if your application displays a confirmation alert after the user chooses a menu command. For example, if the user makes changes to a document, then chooses the Close command, your application can display a confirmation alert box, asking the user whether the document should be saved before closing. This type of command should not contain ellipses in its text.

If your application displays a dialog box requesting more information in response to the choice of a menu command, do include ellipses in the menu item's text. For example, the Open command includes ellipses in its text because the user must provide additional information: the name of the file to open. When you request more information from the user in a dialog box, you should provide an OK button or its equivalent in the dialog box that the user can select to perform the command. The dialog box should also include a Cancel button or its equivalent so that the user can cancel the command. See the chapter "Dialog Manager" in this book for information on creating dialog boxes.

■ A dimmed appearance. When your application disables a menu item, the Menu Manager dims the menu item to indicate that the user can't choose it. Note that the Menu Manager dims the entire menu item, including any mark or icon, the menu text, and any keyboard equivalent symbol. Divider lines always have a dimmed appearance, regardless of whether your application enables them or not. When your application disables an entire menu, the Menu Manager dims the menu title and all menu items in that menu.

Figure 3-5 shows two menus with menu items that illustrate many of the characteristics that you can use when defining your menu items.

Figure 3-5   Two menus with various characteristics
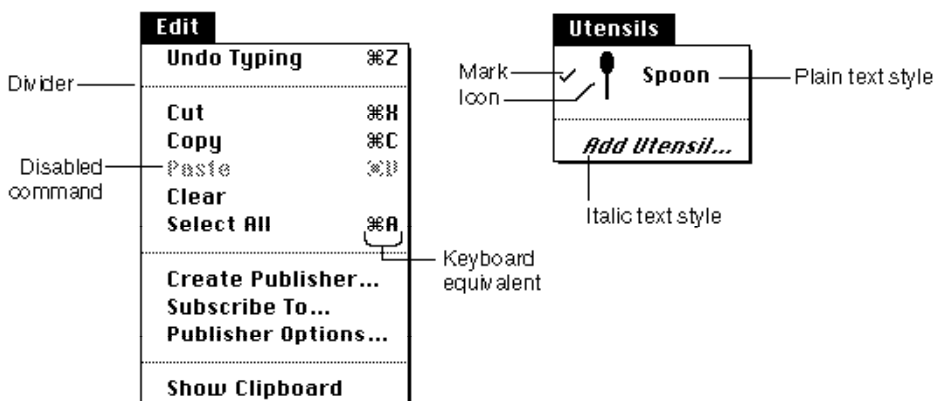
When the primary line direction is right to left (as is the case for non-Roman script systems such as Arabic) the Menu Manager reverses the order of elements in menu items. For example, any marking character appears to the far right and any keyboard equivalent appears to the far left of the menu item's text.

On a monitor that is set to display only black and white, the Menu Manager displays dividers as dotted lines. In all other cases, the Menu Manager displays dividers as appropriate, based on the current color table. For example, on a monitor set to display 4-bit color or greater, the Menu Manager typically displays dividers as gray lines.

Your menu can contain as many menu items as you wish. However, only the first 31 menu items can be individually disabled (all menu items past 31 are always enabled if the menu is enabled and always disabled if the menu is disabled). If your menu items exceed the length of the screen, the user must scroll to view the additional items. Keep in mind that the fewer the menu items in a menu, the simpler and clearer the menu is for the user.

Groups of Menu Items

The menu items in a particular menu should be logically related to the title of the menu and grouped to provide greater ease of use and understanding to the user. You should separate groups with dividers.

A menu can contain both commands that perform actions and commands that set attributes. You should use a verb or verb phrase to name commands that perform actions (for example, Cut, Copy, Paste). You should use an adjective to name commands that set attributes of a selected object (for example, Bold, Italic, Underline). You should group menu items by their type: verbs (actions) or adjectives (attributes). Create groups within each type according to the guidelines described here.

Group action commands that are logically related but independent; this makes your menus easier to read. For example, the Cut, Copy, Paste, Clear, and Select All commands in the Edit menu are grouped together; the Create Publisher, Subscribe To, and Publisher Options commands are grouped together; and the Show Clipboard command is set off by itself. (Figure 3-5 on page 3-13 shows these commands in the Edit menu of a typical application.)

Group attribute commands that are interdependent. You typically group a set of commands that set attributes into either a mutually exclusive group or an accumulating group.

Group a set of attribute commands together if only one attribute in the group can be in effect at any one time (a *mutually exclusive* group). Place a checkmark next to the item that is currently in effect. If the user chooses a different attribute in the group, move the checkmark to the newly chosen attribute. For example, Figure 3-6 shows a Colors menu from the SurfWriter application. The colors listed in the Colors menu form a mutually exclusive group because only one color can be in effect at any one time. In this example, green is the color currently in effect. If the user chooses a different color, such as blue, the SurfWriter application uses the `SetItemMark` procedure to remove the checkmark from the Green command and to place a checkmark next to the Blue command.

Figure 3-6 Menu items in a mutually exclusive group

You can also group a set of attribute commands together if a number of the attributes in the group can be in effect at any one time (an *accumulating* group). In an accumulating group, use checkmarks to indicate that multiple attributes are in effect. In this type of group, you also need to provide a command that cancels all the other attributes. For example, a Style menu that lets the user choose any combination of font styles should also include a Plain Text command that cancels all the other attributes. Figure 3-7 shows a Style menu; in this example, the Bold and Outline attributes are both in effect.

Figure 3-7 Menu items in an accumulating group
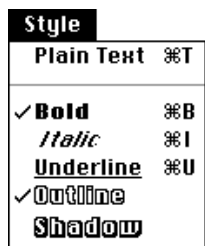
You can also use a combination of checkmarks and dashes to help indicate the state of the user's content. For example, in a menu that reflects the state of a selection, place a checkmark next to an item if the attribute applies to the entire selection; place a dash next to an item if the attribute applies to only part of the selection. Figure 3-8 shows a Style menu that indicates that the selection contains more than one style. In this figure, the Bold attribute applies to the entire selection; the Underline attribute applies to only part of the selection.

Figure 3-8 Use of a checkmark and dash in an accumulating group

Introduction to Menus
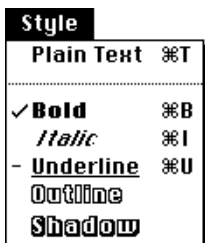
Your application should adjust its menus appropriately before displaying its menus. For example, you should add checkmarks or dashes to items that are attributes as necessary, based on the state of the user's document and according to the type of window that is in the front. See "Adjusting the Menus of an Application" on page 3-73 for more information.

Another way to show the presence or absence of an attribute is to use a toggled command. Use a toggled command if the attribute has two states and you want to allow the user to move between the two states using a single menu command. For example, your application could provide a Show Borders command when the borders surrounding publishers and subscribers are not showing in a document. When the user chooses the Show Borders command, your application should show the borders and change the menu item to Hide Borders. When the user chooses the Hide Borders command, your application should hide the borders surrounding any publishers or subscribers and change the menu item to Show Borders. Use a toggled command only when the wording of the two versions of the command is not confusing to the user. Choose a verb phrase as the text of a toggled command; the text should clearly indicate the action your application performs when the user chooses the item. See "Changing the Text of an Item" on page 3-59 for further information on providing a toggled command.

Keyboard Equivalents for Menu Commands

A menu command can have a keyboard equivalent. The term **keyboard equivalent** refers to a keyboard combination, such as Command-C (⌘-C) or any other combination of the Command key, another key, and one or more modifier keys, that invokes a corresponding menu command when pressed by the user. For example, if your application supports the New command in the File menu, your application should perform the same action when the user presses Command-N as when the user chooses New from the File menu.

The term **Command-key equivalent** refers specifically to a keyboard equivalent that the user invokes by holding down the Command key and pressing another key (other than a modifier key) at the same time. This generates a keyboard event that specifies a 1-byte character that your application should pass as a parameter to the `MenuKey` function. The `MenuKey` function maps the given 1-byte character to the menu item (if any) with that Command-key equivalent.

The Menu Manager provides support for Command-key equivalents. If you define a Command-key equivalent for a menu item, the standard menu definition procedure draws the Command symbol and the specified 1-byte character to the right of the menu item's text (or to the left of the item's text if the primary line direction is right to left).

You detect a Command-key equivalent of a command by examining the `modifiers` field of the event record for a keyboard event. This allows you to determine whether the Command key was pressed at the same time as the keyboard event. If so, your application typically calls the `MenuKey` function, passing as a parameter the character code that represents the key pressed by the user. The `MenuKey` function determines if the 1-byte character matches any of the keyboard equivalents defined for your menu items; if so, `MenuKey` returns this information to your application. Your application can then

perform the associated menu command, if any. See the chapter "Event Manager" in this book for additional information about the `modifiers` field of the event record.

The keyboard layout (`'KCHR'`) resource of some keyboards masks or cancels the effect of the Shift key when the Command key is also pressed. For example, with a U.S. keyboard layout, when a user presses Command-S, the character code in the message field of the event record is $73 (the character code for "s"); when a user presses Command-Shift-S, the character code in the message field of the event record is also $73. However, not all `'KCHR'` resources mask the Shift key in this way.

Furthermore, when your application uses the `MenuKey` function to process Command-key equivalents, `MenuKey` does not distinguish between uppercase and lowercase letters. The `MenuKey` function takes the 1-byte character passed to it and calls the `UpperText` procedure (which provides localizable uppercase conversion of the character). Thus, `MenuKey` translates any lowercase character to uppercase when comparing a keyboard event to keyboard equivalents. If your application must distinguish between lowercase and uppercase characters for keyboard equivalents, you need to provide your own method for handling such keyboard equivalents.

The key you specify for a Command-key equivalent must be a 1-byte character and is usually a letter (although you can specify 1-byte characters other than letters). For consistency and to provide greater support for localizing your application, you should always specify any letters for keyboard equivalents in uppercase when you define your application's menu commands.

If you wish to provide other types of keyboard equivalents in addition to Command-key equivalents, your application must take additional steps to support them. If your application allows the user to hold down more than one modifier key to invoke a keyboard equivalent, your application must provide in the menu item a visual indication that represents this keyboard combination. In most cases your application must use its own method (other than `MenuKey`) for mapping the keyboard equivalent to the corresponding menu item.

If you specify a key other than a letter for a Command-key equivalent or use more than one modifier key for a keyboard equivalent, you should choose keys and keyboard combinations that can be easily localized for other regions.

If your application uses other keyboard equivalents, you can examine the state of the modifier keys and use the `KeyTranslate` function, if necessary, to help map the keyboard equivalent to a particular menu item. See the chapter "Event Manager" in this book for information on the `KeyTranslate` function, and see the discussion of `'KCHR'` resources in *Inside Macintosh: Text* for information on how various keyboard combinations map to specific character codes.

One command that isn't listed in a menu but can be invoked from the keyboard is the Command-period (⌘-.) or Cancel command. You detect a Command-period command in a method similar to the method for detecting other keyboard equivalents—you examine the `modifiers` field of a keyboard event to determine whether the Command key was pressed. In this case, however, if the user pressed the period key in addition to the Command key, rather than invoking a menu command your application should cancel the current operation.

You typically define the Command-key equivalents for your application's menu commands when you define the menu commands in a `'MENU'` resource. The Menu Manager displays the Command-key equivalent for a menu command (if it has one) to the right of the menu item's text (or to the left of the item's text for right-to-left script systems).

Apple reserves several keyboard equivalents for common commands. You should use these keyboard equivalents for commands in the File and Edit menus of your application.

Table 3-1 show the keyboard equivalents for standard commands.

Table 3-1　　Reserved keyboard equivalents for all systems

Keys	Command	Menu
⌘-A	Select All	Edit
⌘-C	Copy	Edit
⌘-N	New	File
⌘-O	Open…	File
⌘-P	Print…	File
⌘-Q	Quit	File
⌘-S	Save	File
⌘-V	Paste	Edit
⌘-W	Close	File
⌘-X	Cut	Edit
⌘-Z	Undo	Edit

Note

You should use the keyboard equivalents Z, X, C, and V for the editing commands Undo, Cut, Copy, and Paste in order to provide support for editing in desk accessories and dialog boxes. ◆

Apple also reserves several keyboard equivalents for use with worldwide versions of system software, localized keyboards, and keyboard layouts. Table 3-2 shows these keyboard equivalents. Your application should not use the keyboard equivalents listed in Table 3-2 for its own menu commands.

See *Inside Macintosh: Text* for more discussion of handling keyboard equivalents in other script systems.

The key combinations listed in Table 3-1 and Table 3-2 are reserved across all applications. Even if your application doesn't support one of these menu commands, it shouldn't use these keyboard equivalents for another command. This guideline is for the user's benefit. Reserving these key combinations provides guaranteed, predictable behavior across all applications.

Table 3-2   Reserved keyboard equivalents for worldwide systems

Keys	Action
⌘–Space bar	Rotate through enabled script systems
⌘–Option–Space bar	Rotate through keyboard layouts or input methods within the active script system
⌘–modifier key–Space bar	Reserved
⌘–Right arrow	Change keyboard layout to the current keyboard layout of the Roman script
⌘–Left arrow	Change keyboard layout to the current keyboard layout of the system script

Table 3-3 shows other common keyboard equivalents. These keyboard equivalents are secondary to the standard keyboard equivalents listed in Table 3-1 and Table 3-2. If your application doesn't support one of the functions in Table 3-3, then you can use the equivalent as you wish.

Table 3-3   Other common keyboard equivalents

Keys	Command	Menu
⌘-B	Bold	Style
⌘-F	Find	File
⌘-G	Find Again	File
⌘-I	Italic	Style
⌘-T	Plain Text	Style
⌘-U	Underline	Style

You shouldn't assign keyboard equivalents to infrequently used menu commands. Only add keyboard equivalents for the commands that your users employ most frequently.

Menus Added Automatically by the Menu Manager

In System 7, the Menu Manager may add as many as three additional menus to your application's menu bar: the Help menu, the Keyboard menu, and the Application menu. These menus provide access to system features such as Balloon Help, keyboard layouts, and application switching. All three of these menus have icons as titles and are positioned at the right side of the menu bar. (These menus are sometimes referred to as the *system-handled menus*.)

The Menu Manager automatically inserts these additional menus in your application's current menu list when your application inserts an Apple menu into its menu bar. In this case, the Menu Manager always displays the Application menu, displays the Help menu if space is available, and displays the Keyboard menu if multiple script systems are installed and space is available. The Menu Manager also displays the Keyboard menu if the smfShowIcon bit is set in the flags byte of the 'itlc' resource.

The Help menu icon or both the Help menu icon and the Keyboard menu icon disappear from the menu bar if your application inserts a menu whose title extends into the space occupied by one or both of those icons. This allows your application to reclaim any space in the menu bar that would have been occupied by one or both of those two menu icons, if necessary. However, if your application inserts a menu whose title is long enough to overlap space occupied by the Application menu icon, the overlapping portion of that title disappears behind the Application menu icon. The Application menu icon is always displayed in the menu bar.

Because the Menu Manager inserts the Help, Keyboard, and Application menus into your application's current menu bar, you should not make any assumptions about the last menu (or menus) in your menu bar. Apple also reserves the right to add other system-handled menus to your application's menu bar; for compatibility you should define your menu bar such that there is room for the Help, Keyboard, and Application menus and at least one additional system-handled menu.

Your application does not need to take any action if the user chooses an item from the Keyboard or Application menu; the Menu Manager performs any appropriate actions for these two menus. If the user chooses an item that your application added to the Help menu, your application should perform the corresponding action.

The following sections describe the Help, Keyboard, and Application menus in more detail, and they also describe other menus in a typical application, including the Apple, File, and Edit menus.

The Apple Menu

You should define the Apple menu as the first menu in your application. The title of the Apple menu is the Apple icon. The Apple menu of an application typically provides an About command as the first menu item, followed by a divider, which is followed by a list of all desktop objects contained in the Apple Menu Items folder. (The phrase *desktop objects* refers to applications, desk accessories, documents, folders, and any other item that can reside in the Apple Menu Items folder.) The items following the divider in the Apple menu are listed in alphabetical order. Each item below the divider lists a desktop object and the small icon for that object.

Figure 3-9 shows the Apple menu for the SurfWriter application as it might appear on a particular user's system.

To create the items in your application's Apple menu, define the Apple menu title, the characteristics of your application's About command, and the divider following it in a `'MENU'` resource.

To insert the items contained in the Apple Menu Items folder into your application's Apple menu, use the `AppendResMenu` or `InsertResMenu` procedure and specify `'DRVR'` as the resource type to add in the parameter `theType`. If you do this, these procedures automatically add all items in the Apple Menu Items folder in alphabetical order to the specified menu.

Figure 3-9 The Apple menu for the SurfWriter application
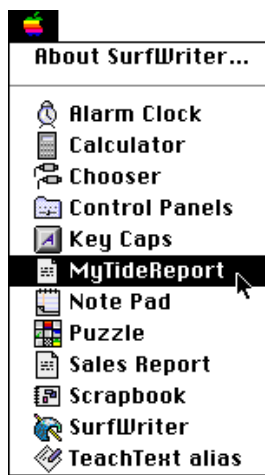
Note

The Apple Menu Items folder is available in System 7 and later. In System 6, the `AppendResMenu` and `InsertResMenu` procedures add only the desk accessories in the System file to the specified menu when you specify `'DRVR'` as the resource type to add in the parameter `theType`. ◆

The user can place any desktop object in the Apple Menu Items folder. When the user places an item in this folder, the system software automatically adds it to the list of items in the Apple menu of all open applications.

When the user chooses an item other than your application's About command from the Apple menu, your application should call the `OpenDeskAcc` function. The `OpenDeskAcc` function prepares to open the desktop object chosen by the user; for example, if the user chooses the Alarm Clock desk accessory, the `OpenDeskAcc` function prepares to open the Alarm Clock. The `OpenDeskAcc` function schedules the Alarm Clock desk accessory for execution and returns to your application. On your application's next call to `WaitNextEvent`, it receives a suspend event, and then the Alarm Clock desk accessory becomes the foreground process.

If the user chooses a desktop object other than a desk accessory or an application, the `OpenDeskAcc` function also takes the appropriate action. For example, as shown in Figure 3-9, if the user chooses a document called MyTideReport created by the SurfWriter application, the `OpenDeskAcc` function prepares to open the SurfWriter application (if it isn't already open) and schedules the SurfWriter application for execution. The SurfWriter application is instructed to open the MyTideReport document when it becomes the foreground process.

When the user chooses your application's About command, your application can display a dialog box or an alert box that contains your application's name, version number, copyright information, or other information as necessary. Your application should provide an OK button in the dialog box; the user clicks the OK button to close the dialog box.

Figure 3-10 shows the alert box that the SurfWriter application displays when the user chooses the About command from the application's Apple menu.

Figure 3-10   Choosing the About command of the SurfWriter application

About SurfWriter...

⏰ Alarm Clock
🖩 Calculator
🖧 Chooser
🖾 Control Panels
🄰 Key Caps
📄 MyTideReport
📝 Note Pad
🔢 Puzzle
📄 Sales Report
📖 Scrapbook
🐦 SurfWriter
🖋 TeachText alias

SurfWriter 3.0
A Swell Text Processor

©My Company, Inc. 1992

OK
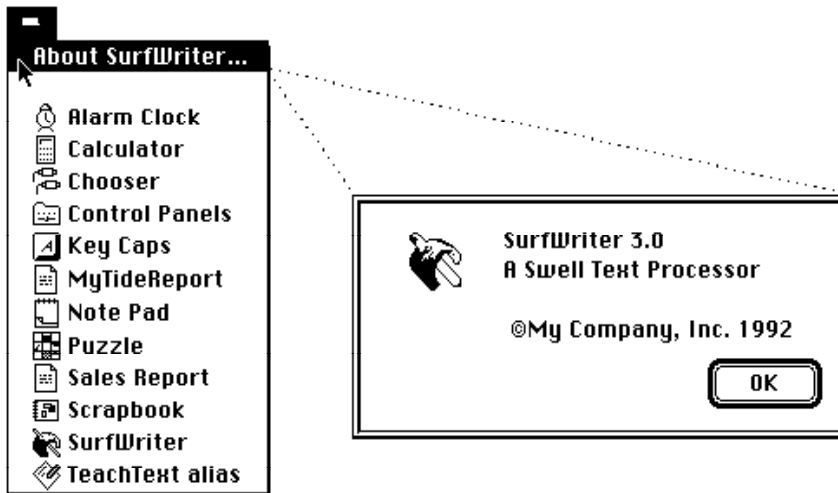
If your application provides any application-specific Help commands, place these in the Help menu, not the Apple menu.

The File Menu

The standard File menu contains commands related to managing documents. For example, the user can open, close, save, or print documents from this menu. The user should also be able to quit your application by choosing Quit from the File menu.

Your application should support the menu commands of the standard File menu. If you add other commands to your application's File menu, they should pertain to managing a document.

Figure 3-11 shows the standard File menu for applications.

Figure 3-11   The standard File menu for an application

File
New   ⌘N
Open...   ⌘O

Close   ⌘W
Save   ⌘S
Save As...

Page Setup...
Print...   ⌘P

Quit   ⌘Q

Table 3-4 describes the standard commands in the File menu and the actions your application should take when a user chooses them.

Table 3-4 Actions for standard File menu commands

Command	Action
New	Open a new, untitled document.
Open...	Display the Open dialog box using the Standard File Package.
Close	Close the active window (which may be a document window, modeless dialog box, or other type of window). If the active window is a document and the document has been changed since the last save, display a dialog box asking the user if the document should be saved before closing.
Save	Save the active document to a disk, including any changes made to that document since the last time it was saved. If the user chooses Save for a new untitled document (one that the user hasn't named yet), display the Save dialog box using the Standard File Package.
Save As...	Save a copy of the active document under a new name provided by the user. Display the Save dialog box using the Standard File Package. After your application saves the document, the document should remain open and active.
Page Setup...	Display the Page Setup dialog box to let the user specify printing parameters such as the paper size and printing orientation. Your application can provide other printing options as appropriate. Your application should save the user's Page Setup printing preferences for the document when the user saves the document.
Print...	Display the Print job dialog box to let the user specify various parameters, such as print quality and number of copies. Print the document if the user clicks the Print button. The options specified in the Print dialog box apply to only the current printing operation, and your application should not save these settings with the document or restore the settings when the user chooses Print again.
Quit	Quit your application after performing any necessary cleanup operations. If any open documents have been changed since the user last saved them, display the Save dialog box once for each open document that requires saving. If any background or lengthy operation is still in progress, notify the user, giving the user the option to continue and not quit the application.

See *Macintosh Human Interface Guidelines* for additional commands that you can provide in the File menu. See the chapter "Introduction to File Management" in *Inside Macintosh: Files* for information on how to perform the actions associated with the commands in the File menu. See the chapter "Standard File Package" in *Inside Macintosh: Files* for information on the standard file dialog boxes. See the chapter "Printing Manager" in *Inside Macintosh: Imaging* for information on displaying the Page Setup and Print job dialog boxes.

The New, Open, Close, Save, Print, and Quit commands have the keyboard equivalents shown in Figure 3-11 on page 3-22. These keyboard equivalents are reserved for these menu commands; do not assign these keyboard equivalents to any menu command other than the ones shown in Figure 3-11.

The Edit Menu

The standard Edit menu provides commands that let users change or edit the contents of their documents. It also provides commands that allow users to share data within and between documents created by different applications using editions or the Clipboard. All Macintosh applications should support the Undo, Cut, Copy, Paste, and Clear commands. Use these commands to provide standard text-editing abilities in your application.

Figure 3-12 shows the standard Edit menu supported by Macintosh applications.

Figure 3-12   The standard Edit menu for an application

The standard editing commands (Undo, Cut, Copy, Paste, and Clear) in your application's Edit menu should appear in the order shown in Figure 3-12. Whenever possible, you should add an additional word or phrase to clarify what action your application will reverse when the user chooses the Undo command. For example, Figure 3-12 shows an application's Edit menu that uses the phrase Undo Typing when typing was the last action performed by the user. If your application can't undo the last operation, you should change the text of the Undo command to Can't Undo and disable the menu item. See "Changing the Text of an Item" on page 3-59 for an example of how to change the text of a menu item.

You can include other commands in your application's Edit menu if they're related to editing or changing the content of your application's documents. If you add commands to the Edit menu, add them after the standard menu commands. For example, if appropriate, your application should support a Select All command. If your application supports both the Clear and Select All commands, they should appear in the order shown in Figure 3-12.

Table 3-5 describes the standard commands in the Edit menu and the actions your
application should take when a user chooses them.

Table 3-5   Actions for standard Edit menu commands

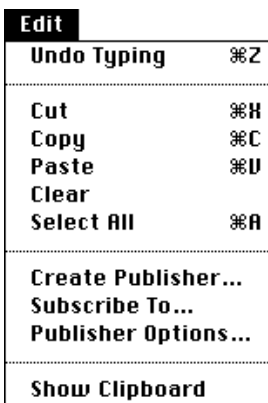| Command	Action
Undo	Reverse the effect of the previous operation. You should add the name of the last operation to the Undo command. For example, change the item to read Undo Typing if the user just finished entering some text in a document. If your application cannot undo the previous operation, disable this menu item and change the phrase to Can't Undo.
Cut	Remove the data in the current selection, if any. Store the cut selection in the scrap (on the Clipboard). This replaces the previous contents of the scrap.
Copy	Copy the data in the current selection, if any. Copy the selection to the scrap (the Clipboard). This replaces the previous contents of the scrap.
Paste	Paste the data from the scrap at the insertion point; this replaces any current selection.
Clear	Remove the data in the highlighted selection; do not copy the data to the scrap (Clipboard).
Select All	Highlight all data in the document.
Create Publisher...	Display the Create Publisher dialog box (using the Edition Manager). Create an edition based on the selected data if the user clicks the Publish button.
Subscribe To...	Display the Subscribe To dialog box (using the Edition Manager). Allow the user to insert data from an edition if the user clicks the Subscribe button.
Publisher Options...	Display the Publisher Options dialog box (using the Edition Manager) and allow the user to set or change options associated with the publisher. Change this menu item to Subscriber Options if the current selection includes a subscriber. When the user chooses the Subscriber Options command, display the Subscriber Options dialog box.
Show Clipboard	Display the contents of the Clipboard in a window. Change this item to Hide Clipboard when the Clipboard window is showing. When the user chooses Hide Clipboard, hide the window displaying the Clipboard contents and change the menu item to Show Clipboard.

The Undo, Cut, Copy, Paste, and Select All commands have the keyboard equivalents
shown in Figure 3-12 on page 3-24. These keyboard equivalents are reserved for these
menu commands; do not assign these keyboard equivalents to any menu command
other than the ones shown in Figure 3-12. See the chapter "Scrap Manager" in *Inside
Macintosh: More Macintosh Toolbox* for information on copying data to and from the scrap.

See the chapter "Edition Manager" in *Inside Macintosh: Interapplication Communication* for information on supporting the Create Publisher, Subscribe To, and Publisher Options commands in your application.

The Font Menu

You can provide a Font menu to allow the user to choose text fonts. A font is a complete set of characters created in one typeface and font style. The characters in a font can appear in many different point sizes, but all have the same design elements.

You should list the names of all currently available fonts in your application's Font menu. The currently available fonts are those fonts residing in the Fonts folder of the user's System Folder (or in earlier versions of system software, in the user's System file).

You add fonts to the Font menu using the `AppendResMenu` or `InsertResMenu` procedure. These two procedures add items to the specified menu in alphabetical order.

The user can install a large number of fonts and thereby create a very large Font menu. Therefore, you should never include other items in the Font menu. Use separate menus to accommodate lists of attributes such as style and size choices. You can also provide a Size menu to allow the user to choose a specific point size of a font; the next section describes the Size menu.

Figure 3-13 shows a typical Font menu. Your application should indicate which typeface is in use by adding a checkmark to the left of the name of the current font. In Figure 3-13, the application has placed a checkmark next to Palatino to indicate that Palatino® is the current font. When the user starts entering text at the insertion point, your application should display text in the current font.

Figure 3-13   A typical Font menu
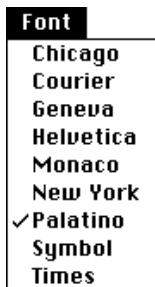
In the Font menu, you can use dashes to indicate that the selection contains more than one font. (Place a checkmark next to an item if the entire selection contains only one font.) If the current selection contains more than one font, place a dash next to the name of each font that the selection contains. See "Changing the Mark of Menu Items" on page 3-61 for information on adding dashes and checkmarks to a menu item.

Figure 3-14 shows the use of dashes to indicate that a selection contains more than one font. In this figure, part of the selection contains a Helvetica® font and part of the selection contains a Palatino font.

Figure 3-14   A Font menu showing a selection containing more than one font

```
┌─────────────────┐
│ Font            │
├─────────────────┤
│  Chicago        │
│  Courier        │
│  Geneva         │
│ - Helvetica     │
│  Monaco         │
│  New York       │
│ - Palatino      │
│  Symbol         │
│  Times          │
└─────────────────┘
```

The `AppendResMenu` and `InsertResMenu` procedures can recognize when an added font resource is associated with a script other than the current system script (non-Roman fonts have font numbers greater than $4000). The Menu Manager displays a font name in its corresponding script if the script system for that font is installed.

You can choose to provide a Size menu and a Style menu in addition to a Font menu. If you do so, these three menus typically appear in the order Font, Size, Style in most applications.

The Size Menu

Your application can provide a Size menu to allow the user to choose sizes for fonts. Font sizes are measured in points. A point is a typographical unit of measure equivalent (on Macintosh computers) to 1/72 of an inch.

Your application should indicate the current point size by adding a checkmark to the menu item of the current size. You can use dashes if the selection contains more than one point size.

System 7 supports both bitmapped and TrueType fonts. TrueType fonts can be displayed in a wider range of point sizes, for example, 12 points, 51 points, 156 points, 578 points, or greater. Your application should not provide an upper limit for font sizes.

In the Size menu, your application should outline font sizes to indicate which sizes are directly provided by the current font. If the user chooses a TrueType font, outline all sizes of that font in the Size menu. If the user chooses a bitmapped font, outline only those sizes that appear in the Fonts folder. Use plain type for all other font sizes. See the chapter "Font Manager" in *Inside Macintosh: Text* for additional information on supporting fonts in your application.

Figure 3-15 shows a typical Size menu of an application.

Figure 3-15   A typical Size menu
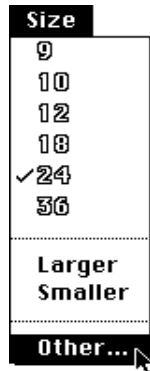
Your application should also provide a method that allows users to choose any point size. You can add an Other command to the end of the Size menu for this purpose. When the user chooses this command, display a dialog box that allows the user to choose any available font size. You can include an editable text item in which the user can type the desired font size. Figure 3-16 shows a dialog box an application might display when the user chooses the Other command from the Size menu.

Figure 3-16   A dialog box to select a new point size for a font
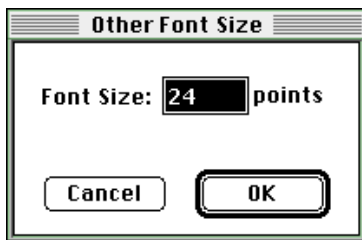
Figure 3-17 shows the Other dialog box after the user has entered a new font size of 31.

Figure 3-17   Entering a new point size for a font
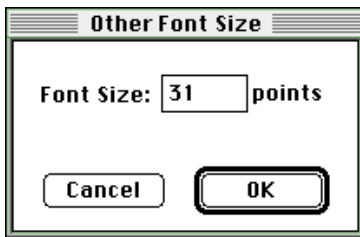
If the user enters a font size not currently in the menu, your application should add a checkmark to the Other menu command and include the font size as part of the text of the Other command. You should show the font size in parentheses after the text Other, as shown in Figure 3-18.

Figure 3-18   The Other command with a font size added to it
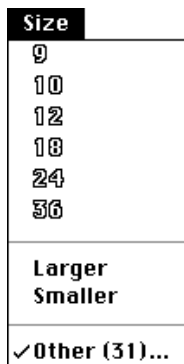
If a selection contains more than one nonstandard size, you should include the text Mixed in parentheses following the word Other. In this case leave the editable text field of the Other dialog box blank when the user chooses the Other (Mixed) command.

See "Handling a Size Menu" on page 3-82 for more information on how to respond to the user's choice of a command from the Size menu. See the chapter "Dialog Manager" for information on creating a dialog box.

The Help Menu

The Help menu is specific to each application, just as the Apple, File, and Edit menus are. The Help menu items defined by the Help Manager are common to all applications and give the user access to Balloon Help.

You can add menu items to your application's Help menu to give your users access to any online help that your application supplies in addition to help balloons. If you currently provide your users with help information when they choose the About

command from the Apple menu, you should instead append a command for your own help to the Help menu. This gives users one consistent place to obtain help information.

When adding your own items to the Help menu, include the name of your application in the command so that users can easily determine which application the help relates to.

Figure 3-19 shows the Help menu for the SurfWriter application. This application appends one item to the end of the standard Help menu: SurfWriter Help. When the user chooses this item, the application provides access to any application-specific help information.

Figure 3-19   The Help menu of the SurfWriter application
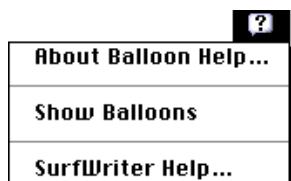
You add items to the Help menu by using the HMGetHelpMenuHandle function and the AppendMenu procedure. Apple reserves the right to change the number of standard items in the Help menu. You should always append any additional items to the end. See "Adding Items to the Help Menu" on page 3-67 for specific examples.

The user turns Balloon Help on or off by choosing Show Balloons or Hide Balloons from the Help menu. The Help Manager automatically enables or disables Balloon Help when the user chooses Show Balloons or Hide Balloons from the Help menu. The setting of help is global and affects all applications.

When the user turns on Balloon Help, the Help Manager displays small help balloons as the user moves the cursor over areas such as scroll bars, buttons, menus, or rectangular areas in windows or dialog boxes that have help information associated with them. **Help balloons** are rounded-rectangle windows that contain explanatory information for the user.

The Help Manager provides help balloons for the menu titles of the Apple, Help, Application, and Keyboard menus. The Help Manager also provides help balloons for menu items in the Application and Keyboard menus, for any item from the Apple Menu Items folder in the Apple menu, and for the standard items in the Help menu. The Help Manager provides these help balloons only if your application uses the standard menu definition procedure.

Your application should provide the content of help balloons for all other menu items and menus in your application.

Figure 3-20 shows the default help balloons for the Apple menu title and Application menu title.

Figure 3-20 Default help balloons for the Apple menu and Application menu

Figure 3-21 shows help balloons for an application's Cut command when it is enabled and when it is disabled.

Figure 3-21 Help balloons for different states of the Cut command
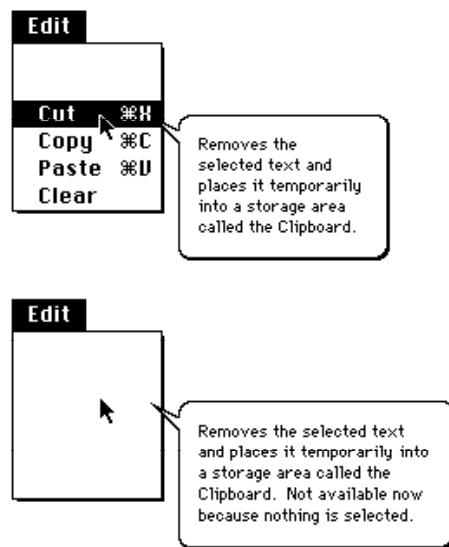
Your application can provide the content for help balloons for your menus and menu items. You define the help balloons for your application using 'hmmu' resources.

For information on how to define the help balloons for your application's menus in 'hmmu' resources, see the chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox*.

Menu Manager

The Keyboard Menu

The Keyboard menu displays a list of all the keyboard layouts and input methods that are available for each enabled script system. Each script system has at least one keyboard layout or input method associated with it. If only the Roman script system and the U.S. keyboard layout are available, the Menu Manager does not add the Keyboard menu (unless the smfShowIcon bit is set in the flags byte of the 'itlc' resource). If the user's system includes an additional script system or includes additional keyboard layouts for the Roman script system and the smfShowIcon bit is set in the 'itlc' resource, the Menu Manager adds the Keyboard menu to your application's menu bar as long as your application's menu bar includes an Apple menu. The Menu Manager adds the Keyboard menu to the right of the Help menu and to the left of the Application menu.

Figure 3-22 shows a Keyboard menu as it might appear on a particular user's system. System software groups the items in the Keyboard menu by their script systems. For example, in Figure 3-22 seven script systems are shown: Arabic, Roman, Cyrillic, Hebrew, Thai, Japanese, and Korean. Two keyboard layouts are available in the user's system for the Arabic script system, two keyboard layouts for the Roman script system, one keyboard layout for the Cyrillic script system, two keyboard layouts for the Hebrew script system, three keyboard layouts for the Thai script system, two input methods for the Japanese script system, and one input method for the Korean script system.

Figure 3-22   Accessing the Keyboard menu from an application
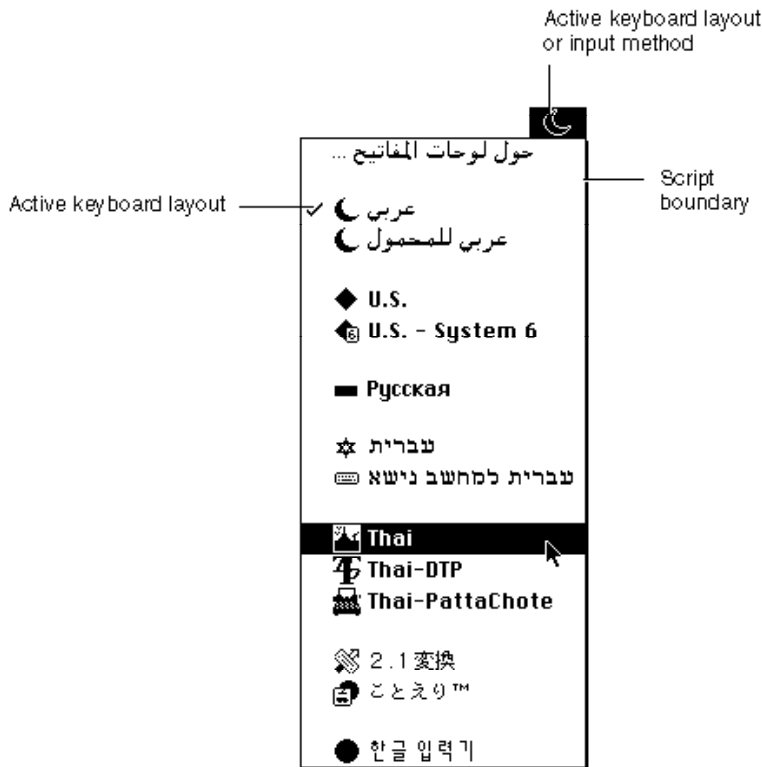
When the user chooses an item from the Keyboard menu, the Menu Manager handles it appropriately. For example, if the user chooses a different keyboard layout in a different script, the Menu Manager changes the current keyboard layout and script system to the item chosen by the user. See *Inside Macintosh: Text* for further information on supporting text and handling text in multiple scripts in your application.

The Application Menu

The Application menu is the menu farthest to the right in the menu bar; the Application menu contains the icon of the active application or desk accessory for its menu title.

The Menu Manager automatically appends the Application menu to your application's menu bar if your menu bar includes an Apple menu.

When the user chooses an item from the Application menu, the Menu Manager handles the event as appropriate. For example, if the user chooses the Hide Others command, the Menu Manager hides the windows of all other open applications. If the user chooses another application from the Application menu, the Menu Manager sends your application a suspend event. Your application receives the suspend event the next time it calls `WaitNextEvent`, and your application is switched out after handling the suspend event. (See the chapter "Event Manager" in this book for information about responding to suspend and resume events.)

Figure 3-23 shows the Application menu for the SurfWriter application as it appears when both SurfWriter and TeachText are open and the user is currently interacting with SurfWriter. The checkmark next to the menu item showing SurfWriter's icon indicates that SurfWriter is the active application.

Figure 3-23   SurfWriter's Application menu

Pop-Up Menus

You can use pop-up menus to present the user with a list of choices in a dialog box or window. Pop-up menus are especially useful in dialog boxes that require the user to select one choice from a list of many or to set a specific value.

In System 7, the standard pop-up menu is implemented by a control definition function. This section explains how the standard pop-up control definition function provides support for pop-up menus. The chapter "Control Manager" in this book explains controls in detail.

Menu Manager

A pop-up menu appears as a rectangle with a one-pixel border and a one-pixel drop shadow. Pop-up menus are identified by a downward-pointing triangle that appears in the pop-up box. The title of the pop-up menu appears next to the pop-up box. Figure 3-24 shows a pop-up menu.

Figure 3-24   A pop-up menu
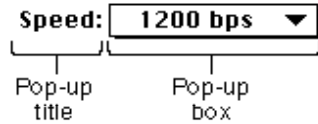
To display a pop-up menu, the user presses the mouse button while the cursor is over the pop-up title or pop-up box. If the pop-up menu is in a dialog box and your application uses the Dialog Manager, the Dialog Manager uses the pop-up control definition function to display the pop-up menu and to handle all user interaction in the pop-up menu. If the pop-up menu is in one of your application's windows, your application needs to determine which control the cursor was in when the user pressed the mouse button. Your application can then use the Control Manager routines to display the pop-up menu and to handle user interaction in the control.

Just like MenuSelect, the pop-up control definition function highlights the pop-up menu title and highlights menu items appropriately as the user drags the cursor through the menu items. The pop-up control definition function also highlights the default (current) menu item when the pop-up menu is first displayed and adds the checkmark to the menu item. Once the user releases the mouse button, the pop-up control definition function causes the chosen item (if any) to blink, unhighlights the menu title, changes the text in the pop-up box, and stores the item number of the chosen item as the value of the control. Your application can use the Control Manager function GetControlValue to get the menu item chosen by the user.

Figure 3-25 shows a pop-up menu in its closed state (as it appears initially to the user) and its open state (as it appears when the user presses the mouse button while the cursor is in the pop-up menu).

Figure 3-25   A pop-up menu in its closed and open states

If you don't provide a title for a pop-up menu, the current menu item serves as the title. In most cases you should create pop-up menus that have titles. Choose a title that reflects the contents of the menu or indicates the purpose of the menu.

Figure 3-26 shows the process of a user making a selection from a pop-up menu.

Figure 3-26 Making a selection from a pop-up menu
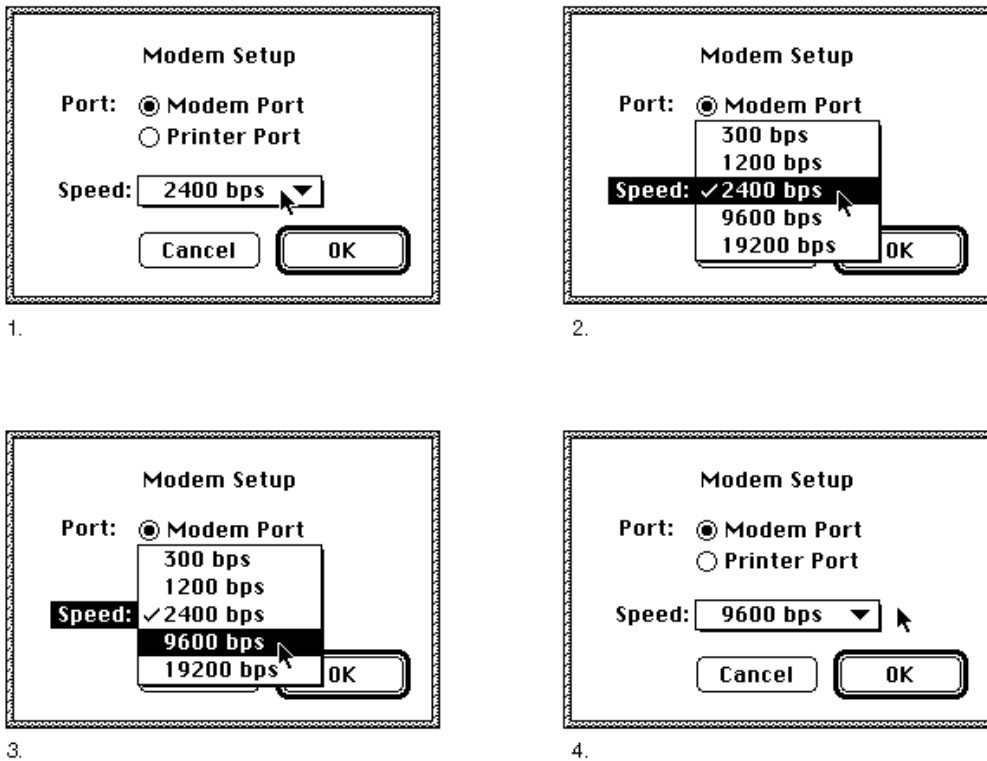
In step 1 in Figure 3-26, the user presses the mouse button while the cursor is over the pop-up box. When this occurs, your application can use the Dialog Manager or Control Manager to call the pop-up control definition function. In step 2, the pop-up control definition function highlights the title of the pop-up menu, removes the downward-pointing triangle from the pop-up box, adds a checkmark to the current item, highlights the current item, and displays the contents of the pop-up menu. In step 3, the pop-up control definition function handles all user interaction, highlighting and unhighlighting menu items, until the user releases the mouse button. When the user releases the mouse button, the pop-up control definition function closes the pop-up menu, unhighlights the pop-up menu title, sets the text of the pop-up box to the item chosen by the user, and stores the item number of the chosen item as the value of the control. Step 4 shows the appearance of the closed pop-up menu after the pop-up control definition function performs these actions.

If your application does not use the standard pop-up control definition function, you can create your own control definition function and you can choose to use the PopUpMenuSelect function to help your application handle pop-up menus. In this case, when the user presses the mouse button when the cursor is in a pop-up menu, your application should call the PopUpMenuSelect function. Your application must

highlight the pop-up title before calling `PopUpMenuSelect` and unhighlight it afterward. The `PopUpMenuSelect` function displays the pop-up menu and highlights menu items appropriately as the user drags the cursor through the menu items. Once the user releases the mouse button, `PopUpMenuSelect` flashes the chosen item, if any, and returns information indicating which menu item was chosen to your application. Your application is responsible for highlighting and unhighlighting the menu title, updating the text in the pop-up box, and storing any changes to the settings of the menu items if you use the `PopUpMenuSelect` function.

Pop-up menus work well when your application needs to present several choices to the user. Note that pop-up menus hide these choices from the user until the user chooses to display the pop-up menu. Use pop-up menus when the user doesn't need to see all the choices all the time. For example, Figure 3-27 shows a dialog box that uses a pop-up menu to allow the user to choose one color from a list of many.

Figure 3-27   Choosing one attribute from a list of many
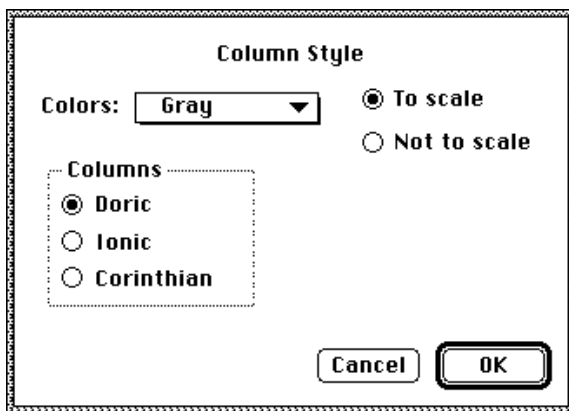
If you need to show only a few choices, you may find that using checkboxes or radio buttons is more appropriate for your application. For example, in Figure 3-27 the selection of columns is implemented with radio buttons rather than a pop-up menu. Whenever possible, you should show all available choices to the user. Note that in this example the amount of space occupied by the radio buttons is about the same as the amount of space required for a corresponding pop-up menu.

Use pop-up menus to allow the user to choose one option from a set of many choices. Don't use a pop-up menu for multiple-choice lists where the user can make more than one selection. If you do, the text in the menu box will not fully describe the selections in effect. For example, don't use a pop-up menu for font style selections. In a dialog box, font style selections are more appropriately implemented as checkboxes. Figure 3-28 shows a dialog box that uses checkboxes instead of a pop-up menu to allow the user to select more than one font style. The Size and Font choices are implemented as pop-up menus in this example, since the user can choose only one size and one font from a list of many.

Figure 3-28 A dialog box with checkboxes and pop-up menus
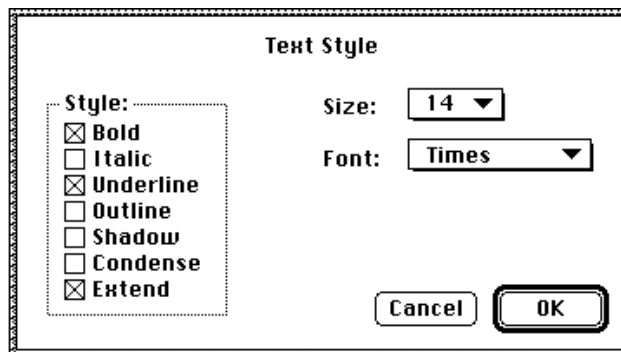
Never use a pop-up menu as a way to provide more commands. Pop-up menus should not contain actions (verbs) but can contain attributes (adjectives) or settings that allow the user to choose one from many. For these reasons, you should not use Command-key equivalents for pop-up menu items.

Your application can also use type-in pop-up menus when appropriate. Use a type-in pop-up menu to give the user a list of choices and to allow the user to type in an additional choice. The standard pop-up control definition function that implements pop-up menus does not provide specific support for type-in menus. You can create your own control definition function to handle type-in pop-up menus. If you do so, your type-in pop-up menu should adhere to the guidelines described here. Figure 3-29 shows a typical type-in pop-up menu in its closed and open states.

Figure 3-29 A type-in pop-up menu in its closed and open states

Your application is responsible for drawing and highlighting the type-in field of the pop-up menu. Your application does not need to highlight the title of a type-in pop-up menu; your application should highlight the type-in field instead.

If the user types in a value that is already in the menu, make that item the current item. If the user types a value that does not match any of the items in the pop-up menu, add the item to the top of the menu and add a divider below the item to separate it from the rest of the standard items. Figure 3-30 on the next page shows a type-in pop-up menu with a user's choice added to it.

Figure 3-30   A type-in pop-up menu with a user's choice added
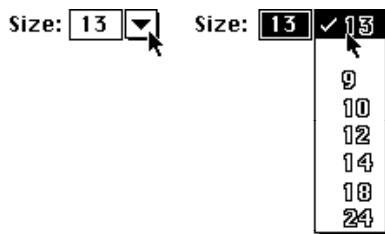
A type-in pop-up menu should allow the user to type in a single additional choice. That is, a standard type-in pop-up menu does not accumulate the user's choices in the menu. For example, if the user types in a value of 13, then types in a new choice, such as 43, the menu should appear as shown in Figure 3-30, except that the type-in field and menu item that previously contained 13 is replaced by 43.

A type-in pop-up menu should also allow the user to type in any of the standard values in the menu or choose any of the standard items in the pop-up menu. If the user types in or chooses any of the standard items, you should remove any user-specified item previously added to the menu. For example, as shown in Figure 3-30, the user specified a nonstandard size of 13. If the user then types in or selects 9, your application should return the pop-up menu to its standard state, as shown in Figure 3-29 on page 3-37.

Hierarchical Menus

A hierarchical menu is a menu that has a submenu attached to it. Hierarchical menus can be useful when your application needs to offer additional choices to the user without taking up extra space in the menu bar. If you use a hierarchical menu in your application, use it to give the user additional choices or to choose attributes, not to choose additional commands.

In a hierarchical menu, a menu item serves as the title of a submenu; this menu item contains a triangle to identify that the item has a submenu. The triangle appears in the location of the keyboard equivalent. The title of a submenu should represent the choices it contains. Figure 3-31 shows a menu with a submenu whose menu title is Label Style.

When a user drags the cursor through a hierarchical menu and rests the cursor on a menu item that has a submenu, the Menu Manager displays the submenu after a brief delay. The title of the submenu remains highlighted while the user browses through the submenu; the Menu Manager unhighlights the menu title of the submenu when the user releases the mouse button.

Hierarchical menus are useful for providing lists of related items, such as font sizes and font styles. Never use more than one level of hierarchical menus (in other words, don't attach a submenu to another submenu). You can assign keyboard equivalents to the menu items of a submenu; however, if you do so, you make it harder for the user to quickly scan all menus for their keyboard equivalents.

Figure 3-31 A hierarchical menu item and its submenu
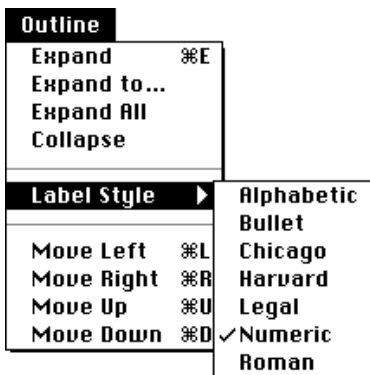
About the Menu Manager

The Menu Manager, together with the menu definition procedure and menu bar definition function, provides your application with a convenient way to manage the menus in your application. The Menu Manager uses two data structures, menu records and menu lists, to manage menus. The next two sections describe how the Menu Manager uses these two data structures. "Using the Menu Manager," which begins on page 3-41, shows how you can use the Menu Manager to

- define a menu using a `'MENU'` resource

- define a menu bar using an `'MBAR'` resource

- install your application's menu bar

- change the appearance of menu items

- add menu items to a menu

- respond to the user when the user chooses a menu item

- handle the Apple and Help menus

- create a pop-up menu

- create a hierarchical menu

- handle access to menus when your application displays a dialog box

- write your own menu definition procedure

How the Menu Manager Maintains Information About Menus

The Menu Manager maintains information about menus in menu records. Each menu record includes certain information about a specific menu, including

■ the menu ID of the menu

■ the horizontal and vertical dimensions of the menu (in pixels)

■ a handle to the menu definition procedure of the menu

■ flags indicating whether each item (for the first 31 items) is enabled or disabled and whether the menu title is enabled or disabled

■ the contents of the menu, including the menu title and other data that defines the menu items

You typically specify most of this information in a menu resource, that is, a resource of type `'MENU'`. When you create a menu, the Menu Manager stores this information in a menu record. A **menu record** is a data structure of type `MenuInfo`. You usually never need to access the information in the menu record directly; the Menu Manager automatically updates the menu record when you make any changes to the menu, such as adding a menu item. See "The Menu Record" beginning on page 3-95 if you need to access the fields of the menu record directly.

The Menu Manager identifies every menu by a number referred to as a **menu ID.** You must assign a menu ID to each menu in your application. Each menu in your application must have a menu ID that is unique from that of any other menu in your application. You can use any number greater than 0 for a menu ID of a pull-down or pop-up menu; submenus of an application can use only menu IDs from 1 through 235; submenus of a desk accessory must use menu IDs from 236 through 255.

When you create a menu, the Menu Manager creates a menu record for the menu and returns a handle to that menu record. To refer to a menu, you usually use either the menu's menu ID or a handle to the menu's menu record.

To refer to a menu item, use the menu item's **item number.** Item numbers identify items in menus; items are assigned item numbers starting with 1 for the first menu item in the menu, 2 for the second menu item in the menu, and so on, up to the number of the last menu item in the menu.

How the Menu Manager Maintains Information About an Application's Menu Bar

A **menu list** contains handles to the menu records of one or more menus (although a menu list can be empty). The end of a menu list can contain handles to the menu records of submenus and pop-up menus; the phrase *submenu portion of the menu list* refers to this portion of the menu list, which contains information about submenus and pop-up menus.

When your application initializes the Menu Manager, the Menu Manager allocates the current menu list, which is initially empty. The contents of the current menu list change as your application adds menus to or removes menus from it.

The **current menu list** contains handles to the menu records of all menus in the current menu bar and the menu records of any submenus or pop-up menus that you have inserted into the current menu list. Your application typically creates a menu list using `GetNewMBar`, and it then sets the current menu list to its newly created menu list using `SetMenuBar`. You can insert other menus in the current menu list using the `GetMenu` function and `InsertMenu` procedure.

The Menu Manager displays the menu bar and the titles of all pull-down menus that are defined in the current menu list when your application calls the `DrawMenuBar` procedure. The Menu Manager displays the menus in the menu bar in the same order that they appear in the current menu list.

The Menu Manager provides routines for adding menus to and removing menus from the current menu list; your application should never access a menu list directly. To refer to a menu list, use the handle returned by `GetNewMBar` or `GetMenuBar`.

The Menu Manager inserts the Help menu, the Keyboard menu if necessary, and the Application menu into your application's menu list if your application calls the `GetNewMBar` function and your menu bar includes an Apple menu; your application then uses `SetMenuBar` to set the current menu list to the newly created menu list. The Menu Manager also inserts these menus into your application's current menu list if your application inserts the Apple menu into the current menu list using the `InsertMenu` procedure. Therefore, you should not make any assumptions about the last menu (or menus) in your application's current menu list.

When your application inserts a submenu into the current menu list, the Menu Manager stores a handle to the menu record of the submenu in the submenu portion of the current menu list. Similarly, when your application inserts a pop-up menu into the current menu list, the Menu Manager stores a handle to the menu record of the pop-up menu in the submenu portion of the current menu list.

Using the Menu Manager

You can define your application's menus and menu bar as resources and use Menu Manager routines to create and manage them. For example, whenever the user presses the mouse button while the cursor is in the menu bar, your application should call the `MenuSelect` function, allowing the user to choose a command from any menu. The `MenuSelect` function handles all user activity until the user releases the mouse button. The `MenuSelect` function displays and removes menus as the user drags the cursor through the menu bar, and it highlights enabled menu items as the user drags through a menu.

You should provide help balloons for each menu title and menu item of your application. You store information and text for help balloons in resources. See the chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox* for complete and specific information on how to provide help balloons for the menus of your application. The BalloonWriter application, available from APDA, can also help you create help balloons for the menus of your application.

Your application needs to initialize QuickDraw, the Font Manager, and the Window Manager before using the Menu Manager. Your application can accomplish this using the `InitGraf`, `InitFonts`, and `InitWindows` procedures. To initialize the Menu Manager, use the `InitMenus` procedure.

If your application uses pop-up menus, use the `Gestalt` function with the `gestaltPopUpAttr` selector to determine if the control definition function for pop-up menus is available. See *Inside Macintosh: Operating System Utilities* for information about the `Gestalt` function.

To create the pull-down menus in your application's menu bar, you need to

■ create descriptions of each pull-down menu in `'MENU'` resources

■ create an `'MBAR'` resource that lists the order and resource ID of each menu

■ use the `GetNewMBar` function and `SetMenuBar` procedure to set up your menu bar and use the `DrawMenuBar` procedure to draw your menu bar

The next section, "Creating a Menu," explains these steps in detail.

After creating your application's menu bar, you can enable or disable your menu items, add marks such as checkmarks or dashes to menu items, or add items to any of your menus as needed. See "Enabling and Disabling Menu Items" on page 3-58, "Changing the Mark of Menu Items" on page 3-61, and "Adding Items to a Menu" beginning on page 3-64 for information on these topics.

"Handling User Choice of a Menu Command," beginning on page 3-70, shows how to handle mouse-down events in the menu bar, adjust the menus of your application, and determine if the user chose a keyboard equivalent of a command.

"Responding When the User Chooses a Menu Item," beginning on page 3-78, describes how your application should respond once the user chooses an item and also shows how to handle the user's choice of a command from the Apple and Help menus.

If your application displays dialog boxes, see "Accessing Menus From a Dialog Box" beginning on page 3-84.

Finally, if your application needs to create submenus or pop-up menus, see "Creating a Hierarchical Menu" on page 3-53 and "Creating a Pop-Up Menu" on page 3-56.

Creating a Menu

You use various Menu Manager routines to set up the menus and the menu bar for your application. You can use any of these methods to create pull-down menus for your application:

■ You can create descriptions of your application's menus in `'MENU'` resources and describe your application's menu bar in an `'MBAR'` resource. You use the `GetNewMBar` function to read in descriptions of your menu bar and menus and create a new menu list, use the `SetMenuBar` procedure to set the current menu list to your application's menu list, and use the `DrawMenuBar` procedure to update the menu bar.

■ You can create descriptions of your application's menus in `'MENU'` resources, read them in using `GetMenu`, add them to the current menu list using `InsertMenu`, and update the menu bar using `DrawMenuBar`.

■ You can use `NewMenu` to create new empty menus; use `AppendMenu`, `InsertMenuItem`, `InsertResMenu`, or `AppendResMenu` to fill the menus with menu items; add the menus to the current menu list using `InsertMenu`; and update the menu bar using `DrawMenuBar`.

Whenever possible you should define your menus in menu (`'MENU'`) resources and your menu bar in a menu bar (`'MBAR'`) resource to make your application easier to localize.

To create a hierarchical menu, you need to create descriptions of the submenu and the menu to which the submenu is attached. Usually you create the description of both menus in `'MENU'` resources. You typically read in the description of the hierarchical menu using `GetNewMBar` (if you also provide an `'MBAR'` resource). To read in the description of the submenu and insert it in the current menu list, use the `GetMenu` function and `InsertMenu` procedure.

To create a pop-up menu, create descriptions of the pop-up menu and its menu items, create a control that uses the pop-up control definition function, and associate the control with a window or dialog box. You can display and manage the pop-up menu using the Dialog Manager or Control Manager routines.

Once the Menu Manager creates a menu for your application, if necessary you can add additional menu items to the menu using `AppendMenu`, `InsertMenuItem`, `InsertResMenu`, or `AppendResMenu`. You can use various Menu Manager routines to change the appearance of menu items.

The next sections describe how to create `'MENU'` and `'MBAR'` resources. "Creating a Hierarchical Menu" on page 3-53 describes how to create a menu that has a submenu, and "Creating a Pop-Up Menu" on page 3-56 describes how to create pop-up menus.

Creating a Menu Resource

Usually you should define your menus in menu (`'MENU'`) resources so that you can easily localize the menu titles and menu items for other languages, cultures, or regions. A `'MENU'` resource defines the menu title of a menu and the characteristics of menu items in a menu. Listing 3-1 shows a sample `'MENU'` resource in Rez format for an application's Apple menu. (Rez is a resource compiler available with MPW. You can also define menus using a resource utility, such as ResEdit, available from APDA.)

Listing 3-1   Rez input for a `'MENU'` resource for the Apple menu

```
#define mApple 128

resource 'MENU' (mApple, preload) { /*resource ID, preload resource*/
      mApple,                       /*menu ID*/
      textMenuProc,                 /*uses standard menu definition */
                                    /* procedure*/
      0b111111111111111111111111111101,  /*enable About item, */
                                    /* disable divider, */
                                    /* enable all other items*/
```

```
enabled,                            /*enable menu title*/
 apple,                             /*menu title*/
{
                                    /*first menu item*/
      "About SurfWriter…",            /*text of menu item */
                                    /* (includes ellipsis)*/
                                    /*item characteristics follow*/
            noicon,                 /*icon number (if any) or */
                                    /* script code (if any)*/
            nokey,                  /*keyboard equivalent (if any) */
                                    /* or submenu (if any) or */
                                    /* small or reduced icon (if any)*/
            nomark,                 /*marking character (if any) or */
                                    /* menu ID of submenu (if any)*/
            plain;                  /*style of menu item text*/
                                  /*second menu item*/
       "-",                         /*item text (divider)*/
            noicon, nokey, nomark, plain  /*item characteristics*/
}
};
```

You should also define help balloons for each of your application's menu items and each menu title when you create your menus. (Figure 3-21 on page 3-31 shows help balloons for an application's Cut command.) You define the help balloons for your application's menus in `'hmmu'` resources. See the chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox* for examples of how to create `'hmmu'` resources.

Listing 3-1 defines the resource ID of the Apple menu as 128. You can use any number equal to or greater than 128 as a resource ID for a menu. By convention, many applications use 128 as the resource ID of the first menu in the application's menu bar (the Apple menu) and use sequential numbers for the resource IDs of following menus.

Listing 3-1 also defines the menu ID of the Apple menu as 128. Once your application creates the menu, the Menu Manager uses the defined menu ID to refer to this menu. The number you define for the menu ID of a menu does not have to match the resource ID of the menu, but it is usually more convenient to use the same number. You can use any number greater than 0 for the menu ID of a pull-down or pop-up menu; submenus of an application can only use menu IDs from 1 through 235; submenus of a desk accessory must use menu IDs from 236 through 255.

The listing specifies that this menu uses the standard menu definition procedure. If you choose to create your own menu definition procedure, list its resource ID instead of the `textMenuProc` constant.

After the resource ID of the menu definition procedure is a 32-bit number (expressed as a 31-bit field followed by a `Boolean` field), where bits 1–31 indicate if the corresponding menu item is disabled or enabled, and bit 0 indicates whether the menu is enabled or disabled.

The listing specifies in the 31-bit field that the first menu item should be enabled, that the second menu item should be disabled, and that the following menu items (item numbers 3 through 31) should be enabled when the menu is first created. After creating a menu, your application can enable or disable menu items using the `EnableItem` or `DisableItem` procedure. If a menu contains more than 31 items, the Menu Manager automatically enables all items following the 31st item when the menu is enabled. Your application cannot disable any individual items following the 31st item. However, you can disable all items, including items after the 31st item, by disabling the entire menu.

Listing 3-1 specifies that the menu title should be enabled when it is first created. Your application can also disable or enable the menu title using the `DisableItem` or `EnableItem` procedure. When you disable a menu using the `DisableItem` procedure, the Menu Manager disables all menu items in the menu (including any items following the 31st item) and dims the title of the menu.

The resource listing identifies the title of the menu using the constant `apple`. If you specify the `apple` constant as the title, the Menu Manager uses a small Apple icon as the title of the menu. The Menu Manager uses a color Apple icon if the monitor is set to display colors. The listing then defines the characteristics of each menu item in the menu. For each menu item, you need to define the text and any other characteristics of the menu item. For example, Listing 3-1 defines the first item in the Apple menu as the About command; note that the text of this menu item specifies three ellipsis points (...). Specify three ellipsis points following the text of a menu command if your application displays a dialog box requesting information from the user before performing the command. In general, you should not use ellipses if your application displays a confirmation alert after the user chooses a menu command; the About command is an exception to this guideline.

Listing 3-1 defines other characteristics of the About command—it doesn't have an icon to the left of the menu item text, it doesn't have a keyboard equivalent, it doesn't have any mark to the left of the menu item text, and the font style of the menu item text is plain.

By specifying various combinations of values in the icon field and keyboard equivalent field, you can define an icon (normal, small, reduced, or color), a keyboard equivalent, a submenu, or the script code of a menu item. Note that some characteristics are mutually exclusive (for example, an item can have a keyboard equivalent or submenu, but not both), as described in the following paragraphs. Table 3-6 on page 3-46 summarizes how the Menu Manager interprets these item characteristics.

Table 3-6   Specifying submenus, script codes, reduced icons, small icons, and color icons of
                   a menu item in a menu resource

Keyboard equivalent field	Icon field	Marking character field	Description
$1B		Menu ID of submenu	Indicates the item has a submenu. The marking character field specifies the menu ID of the submenu.
$1C	Script code of item text		Indicates the item text uses the script defined by the script code specified in the icon field.
$1D	Icon number of 'ICON' resource		Indicates the item has an icon defined by an 'ICON' resource and that it should be reduced to fit in a 16-by-16 bit rectangle.
$1E	Icon number of 'SICN' resource		Indicates the item has an icon defined by an 'SICN' resource.
$00 or >$20	Icon number of 'ICON' or 'cicn' resource		Indicates the item has an icon defined by an 'ICON' or a 'cicn' resource. (A value greater than $20 in the keyboard equivalent field specifies the item's keyboard equivalent.)

To assign an icon to a menu item, specify an icon number in place of the noicon constant. The icon number you specify should be a number from 1 through 255 (or from 1 through 254 for small icons and reduced icons); add 256 to your icon number and use the result for the resource ID of the color icon ('cicn') resource, icon ('ICON') resource, or small icon ('SICN') resource that describes the icons for the menu item. You must define the icon for a menu item in a 'cicn', an 'ICON', or an 'SICN' resource; the Menu Manager uses only these types of resources for icons you define for your menu items. The Menu Manager first looks for a 'cicn' resource with the calculated resource ID and uses that icon if it finds it. If it doesn't find a 'cicn' resource (or if the computer doesn't have Color QuickDraw) and the keyboard equivalent field specifies $1E, the Menu Manager looks for an 'SICN' resource with the calculated resource ID. Otherwise, the Menu Manager looks for an 'ICON' resource and plots it in a 32-by-32 bit rectangle, unless the keyboard equivalent field contains $1D. If the keyboard equivalent field contains $1D, the Menu Manager reduces the icon to fit in a 16-by-16 bit rectangle.

If you provide an 'ICON' resource and specify the nokey constant or a value greater than $20 as the keyboard equivalent, the Menu Manager enlarges the rectangle of the entire menu item to fit the 32-by-32 bit 'ICON' resource. If you specify a value of $1D as the keyboard equivalent of the menu item, the Menu Manager reduces the 'ICON' resource to fit in a 16-by-16 bit rectangle. If you provide an 'SICN' resource and specify a value of $1E as the keyboard equivalent of a menu item, the Menu Manager plots the small icon in a 16-by-16 bit rectangle. If you provide a 'cicn' resource, the Menu Manager automatically enlarges the enclosing rectangle of the menu item according to the rectangle specified in the 'cicn' resource. (For the Apple and Application menus,

the Menu Manager automatically reduces the icon to fit within the enclosing rectangle of a menu item or uses the appropriate icon from the application's icon family, such as an `'ics8'` resource, if one is available.) See the chapter "Finder Interface" in this book for details on how to create icons for your application.

To assign a keyboard equivalent to a menu item, specify the 1-byte character that the user types in addition to the Command key in place of the `nokey` constant in your resource definition for the menu item. If your application attaches a submenu to a menu item, then specify the `hierarchicalMenu` constant in place of the `nokey` constant. A menu item can have either a keyboard equivalent or submenu defined for it, but not both. To indicate that a menu item has an icon that is defined in an `'SICN'` resource, specify $1E in place of the `nokey` constant. To indicate that a menu item has an icon that is defined in an `'ICON'` resource and that the Menu Manager should reduce this icon to a 16-by-16 bit rectangle, specify $1D in place of the `nokey` constant. Menu items that have small icons or reduced icons cannot have keyboard equivalents.

To set the script code of a menu item's text, specify $1C in place of the `nokey` constant and define the desired script code in place of the `noicon` constant. If an item contains $1C in its keyboard equivalent field and a script code in its icon field, the Menu Manager draws the item's text in the script identified by the script code value if the corresponding script system is installed. If you do not specify a script code for a menu item, the Menu Manager displays the menu item's text in the system font of the current system script. For Roman scripts, the system font is Chicago and the system font size is 12.

To assign a mark that appears to the left of the menu item text and to the left of any icon, specify the marking character in place of the `nomark` constant in your resource definition. If the menu item has a submenu, then specify the menu ID of the submenu in place of the `nomark` constant. Submenus of an application must use menu IDs from 1 through 235; submenus of a desk accessory must use menu IDs from 236 through 255. Note that defining the menu ID of a submenu in a `'MENU'` resource does not attach the submenu to its menu. You must use the `GetMenu` function and `InsertMenu` procedure to do this. "Creating a Hierarchical Menu," which begins on page 3-53, gives information on attaching a submenu to its menu.

To assign a font style to a menu item, in your `'MENU'` resource use the constants `bold`, `italic`, `plain`, `outline`, and `shadow` to get their corresponding styles.

Listing 3-1 defines the second menu item as a divider. When you use a hyphen as the first character in the string that defines the text of a menu item, the Menu Manager creates a divider that extends across the entire width of the menu item. You cannot assign any other characteristics to a divider.

The `'MENU'` resource for the Apple menu does not list any other menu items. Use the `AppendResMenu` procedure to add the desktop items to the Apple menu after your application creates the menu. See "Adding Items to the Apple Menu" on page 3-68 for more information.

Once you create a menu, you can append additional items to it using the `AppendMenu`, `InsertMenuItem`, `InsertResMenu`, or `AppendResMenu` procedure. You can also change the characteristics of individual menu items using Menu Manager routines. See "Changing the Appearance of Items in a Menu" on page 3-57 for more information.

Figure 3-12 on page 3-24 shows a typical Edit menu for an application. Listing 3-2 shows a `'MENU'` resource for this Edit menu.

Listing 3-2   Rez input for a `'MENU'` resource for an Edit menu

```
#define mEdit 130
resource 'MENU' (mEdit, preload) {      /*resource ID, preload resource*/
    mEdit,                              /*menu ID*/
    textMenuProc,                       /*uses standard menu definition */
                                        /* procedure*/
    0b0000000000000000001001000000000,  /*enable/disable first 31 menu */
                                        /* items as appropriate*/
    enabled,                            /*enable title*/
    "Edit",                             /*text of menu title*/
     {                                  /*menu items*/
    "Undo",  noicon, "Z",   nomark, plain; /*keyboard equivalent Command-Z*/
    "-",     noicon, nokey, nomark, plain;
    "Cut",   noicon, "X",   nomark, plain; /*keyboard equivalent Command-X*/
    "Copy",  noicon, "C",   nomark, plain; /*keyboard equivalent Command-C*/
    "Paste", noicon, "V",   nomark, plain; /*keyboard equivalent Command-V*/
    "Clear", noicon, nokey, nomark, plain;
    "Select All",
             noicon, "A",   nomark, plain; /*keyboard equivalent Command-A*/
    "-",     noicon, nokey, nomark, plain;
    "Create Publisher…",
             noicon, nokey, nomark, plain;
    "Subscribe To…",
             noicon, nokey, nomark, plain;
    "Publisher Options…",
             noicon, nokey, nomark, plain;
    "-",     noicon, nokey, nomark, plain;
    "Show Clipboard",
             noicon, nokey, nomark, plain
     }
};
```

Listing 3-2 defines the resource ID of the Edit menu as 130, defines the menu ID of the Edit menu as 130, and specifies that this menu uses the standard menu definition procedure. The listing defines the initial enabled state of the first 31 menu items and also specifies that the menu title should be enabled when it is first created.

The resource listing defines the title of the menu, Edit. It then defines the characteristics of each menu item in the menu. For each menu item, you need to specify the text of the menu item and any other characteristics of the menu item. For example, Listing 3-2

defines the first item in the Edit menu as the Undo command with these characteristics: there is no icon to the left of the menu item text, the menu item has a keyboard equivalent of Command-Z, it does not have any mark to the left of the menu item text, and the style of the menu item text is plain. The listing defines the second menu item as a divider line. It defines the Cut, Copy, and Paste commands; specifies keyboard equivalents for each of them; and defines the rest of the items in the menu.

Listing 3-3 shows another example of a resource description of a menu, the File menu of a typical application.

Listing 3-3   Rez input for a `'MENU'` resource for a File menu

```
resource 'MENU' (mFile, preload) {
     mFile, textMenuProc,
     0b0000000000000000000010000000000,
     enabled,
     "File",
     {
         "New",          noicon, "N",   nomark, plain;
         "Open…",        noicon, "O",   nomark, plain;
         "-",            noicon, nokey, nomark, plain;
         "Close",        noicon, "W",   nomark, plain;
         "Save",         noicon, "S",   nomark, plain;
         "Save As…",     noicon, nokey, nomark, plain;
         "-",            noicon, nokey, nomark, plain;
         "Page Setup…", noicon, nokey, nomark, plain;
         "Print…",       noicon, "P",   nomark, plain;
         "-",            noicon, nokey, nomark, plain;
         "Quit",         noicon, "Q",   nomark, plain
     }
};
```

Creating a Menu Bar Resource

You typically define your application's menu bar using a menu bar (`'MBAR'`) resource. Listing 3-4 shows an `'MBAR'` resource, in Rez format, for a sample application.

Listing 3-4   Rez input for an `'MBAR'` resource

```
#define rMenuBar      128

#define mApple        128
#define mFile         129
#define mEdit         130
```

```
resource 'MBAR' (rMenuBar, preload) {/*resource ID, preload*/
   /*menus appear in the order listed here*/
   { mApple, mFile, mEdit };          /*resource IDs for menus in */
                                      /* this menu bar*/
};
```

Listing 3-4 defines the `'MBAR'` resource with resource ID 128. This `'MBAR'` resource defines the order and resource IDs of the menus contained in it; it defines its first three menus as the menus with resource IDs 128, 129, and 130. The Menu Manager uses the assigned resource IDs to read in the menus when it creates a menu bar from an `'MBAR'` resource.

Setting Up Your Application's Menu Bar

To create a menu list as defined in an `'MBAR'` resource, use the `GetNewMBar` function. For each menu defined by the `'MBAR'` resource, the `GetNewMBar` function creates a menu record for the menu, creates each menu according to its resource definition in its corresponding `'MENU'` resource, and inserts each menu into the new menu list. The `GetNewMBar` function returns a handle to the created menu list. For example, this code creates a menu list for the menu bar defined by the `'MBAR'` resource with resource ID 128 (defined by the constant `rMenuBar`):

```
CONST
   rMenuBar = 128;
VAR
   menuBar:    Handle;

 menuBar := GetNewMBar(rMenuBar); {read menus and menu bar }
                                  { descriptions,create & return }
                                  { a handle to a new menu list}
```

Use the `SetMenuBar` procedure to set the current menu list to the menu list created by your application and the `DrawMenuBar` procedure to update the menu bar's appearance. For example, Listing 3-5 uses these two routines to set up the application's menu bar.

Listing 3-5   Setting up an application's menus and menu bar

```
PROCEDURE MyMakeMenus;
VAR
   menuBar:                 Handle;
 BEGIN
{first use the GetNewMBar function to read menus in & create a }
{ new menu list. If you define an Apple menu, the Menu Manager }
{ inserts the Help and Application menus (and Keyboard menu if }
{ necessary) into the newly created menu list}
```

```
    menuBar := GetNewMBar(rMenuBar);
    IF menuBar = NIL THEN
        EXIT(MyMakeMenus);
    SetMenuBar(menuBar); {insert menus into the current menu list}
    DisposHandle(menuBar);
                        {add desktop items in Apple Menu Items }
                        { folder to Apple menu}
    AppendResMenu(GetMenuHandle(mApple), 'DRVR');
    MyAdjustMenus;          {adjust items and enabled state of menus}
    DrawMenuBar;            {draw the menu bar}
END;
```

The code in Listing 3-5 creates the application's menu bar by reading in the definition from the `'MBAR'` resource with resource ID 128, and it uses `SetMenuBar` to set the current menu list to the newly created menu list. The code then adds the desktop items in the Apple Menu Items folder to the Apple menu using the `AppendResMenu` procedure.

You can use the `GetMenuHandle` function to get a handle to the menu record of any menu in the current menu list. You supply the menu ID of the desired menu as a parameter to `GetMenuHandle`, and `GetMenuHandle` returns a handle to the menu's menu record. Most Menu Manager routines require either a menu ID or a handle to a menu record as a parameter.

After creating the menu bar and adding any other menus or items as necessary, the code calls the `MyAdjustMenus` procedure to adjust the application's menus—for example, this procedure sets the enabled and disabled states of menu items in accordance with the current state of the application. (Listing 3-19 on page 3-74 shows the application-defined `MyAdjustMenus` procedure used in Listing 3-5.) After adjusting the menus, the code in Listing 3-5 uses `DrawMenuBar` to draw the menus in the menu bar according to their current enabled state and as they are defined in the current menu list.

Usually you'll define the menus of your application and its menu bar using `'MENU'` resources and an `'MBAR'` resource and using the `GetNewMBar` function to read the resource definitions. However, you can choose to read in a `'MENU'` resource using the `GetMenu` function or to create a new empty menu using `NewMenu`. You can then insert a menu into the current menu list using the `InsertMenu` procedure. See "Creating Menus" on page 3-105 and "Adding Menus to and Removing Menus From the Current Menu List" on page 3-108 for information on forming your menus using these routines.

If your application uses a submenu, you need to use the `GetMenu` function and `InsertMenu` procedure to make these menus available to your application. See "Creating a Hierarchical Menu" on page 3-53 for information on creating submenus. If your application uses a pop-up menu, you can use the pop-up control definition function and Dialog Manager or Control Manager routines to create and display the pop-up menu. See "Creating a Pop-Up Menu" on page 3-56 for information on creating pop-up menus.

The Menu Manager creates and initializes your application's menu color information table when your application calls `GetNewMBar`. You can add entries to your application's menu color information table if you want to use colors other than the default colors in your menus and menu bar. You can add entries to this table by providing menu color information table (`'mctb'`) resources or by using the `SetMCEntries` procedure. Usually you should use the default colors to help maintain a consistent user interface.

If you add menu color entries to your application's menu color information table and your application uses more than one menu bar, you need to save a copy of your application's menu color information table before changing menu bars. Use the `GetMCInfo` function before calling `GetNewMBar` and call `SetMCInfo` afterward to restore the menu color information table. Listing 3-6 shows a routine that saves and then restores the menu color information table when creating a new menu bar.

Listing 3-6   Saving and restoring menu color information

```
PROCEDURE MyChangeMenuBarAndSaveColorInfo;
CONST
   rMenuBar2 = 129;
VAR
   menu:                MenuHandle;
   menuBar:             Handle;
   currentMCTable:      MCTableHandle;
   newMCTable:          MCTableHandle;
 BEGIN
   currentMCTable := GetMCInfo;      {save menu color info table}
   IF currentMCTable = NIL THEN
      EXIT(MyChangeMenuBarAndSaveColorInfo);
   menuBar := GetNewMBar(rMenuBar2);{read menus in & create new menu list}
   IF menuBar = NIL THEN
      EXIT(MyChangeMenuBarAndSaveColorInfo);
   newMCTable := GetMCInfo;          {get new menu color info table}
   IF newMCTable = NIL THEN
      EXIT(MyChangeMenuBarAndSaveColorInfo);
   SetMCInfo(currentMCTable);        {restore previous menu color info table}
   SetMenuBar(menuBar);              {insert menus into the current menu list}
   DisposHandle(menuBar);
   AppendResMenu(GetMenuHandle(m2Apple), 'DRVR'); {add desktop items from }
                                     { Apple Menu Items folder to Apple menu}
   MyAdjustMenus;                          {adjust menu items}
   DrawMenuBar;                            {draw the menu bar}
 END;
```

Creating a Hierarchical Menu

A hierarchical menu is a menu that has a submenu attached to one or more of its menu items. Submenus can be useful when your application needs to offer additional choices to the user without taking up extra space in the menu bar. If you use a submenu in your application, use it to give the user additional choices or to choose attributes, not additional commands.

A menu item of a pull-down menu is the title of the attached submenu. A menu item that has a triangle facing right in the location of the keyboard equivalent identifies that a submenu is attached to the menu item. The title of a submenu should represent the choices it contains. Figure 3-32 shows a menu with a submenu whose menu title is Label Style.

Figure 3-32   A menu item with a submenu
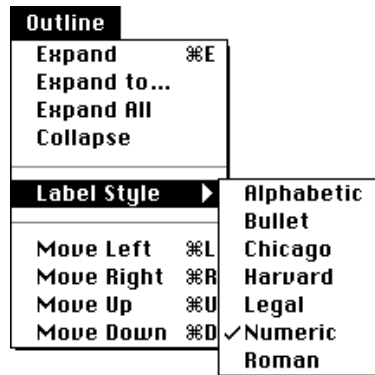
When a user drags the cursor through a menu and rests it on a menu item with a submenu attached to it, the Menu Manager displays the submenu after a brief delay. The title of the submenu remains highlighted while the user browses through the submenu; the Menu Manager unhighlights the menu title of the submenu when the user releases the mouse button.

Your application is responsible for placing any marks next to the current choice or attribute of the submenu. For example, in Figure 3-32 the application placed the checkmark next to the Numeric menu item to indicate the current choice. If the user makes a new choice from the menu, your application should update the menu items accordingly.

You can specify that a particular menu item has a submenu by identifying this characteristic (using the `hierarchicalMenu` constant) when you define the menu item in its `'MENU'` resource. You cannot assign keyboard equivalents to a menu item that has a submenu. (You can define keyboard equivalents for the menu items in the submenu, but this is not recommended.) You identify the menu ID of the submenu in place of the marking character in the menu item's resource description. Thus, a menu item that has a submenu cannot have a marking character and cannot have a keyboard equivalent.

To insert a submenu into the current menu list, you must use the `InsertMenu` procedure. The `GetNewMBar` function does not read in the resource descriptions of any submenus.

Listing 3-7 shows the `'MENU'` resource for an application-defined menu called Outline. The Outline menu contains a number of menu items, including the Label Style menu item. The description of this menu item contains the constant `hierarchicalMenu`, which indicates that the item has a submenu. This menu item description also contains the menu ID of the submenu (defined by the `mSubMenu` constant). The menu ID of a submenu of an application must be from 1 through 235; the menu ID of a submenu of a desk accessory must be from 236 through 255.

The submenu is defined by the menu with the menu ID specified by the Label Style menu item. You define the menu items of a submenu in the same way as you would a pull-down menu (except you shouldn't define keyboard equivalents for items in a submenu, and you shouldn't attach a submenu to another submenu).

Listing 3-7 Rez input for a description of a hierarchical menu with a submenu

```
#define mOutline 135
#define mSubMenu 181

resource 'MENU' (mOutline, preload) {
    mOutline ,                          /*menu ID*/
    textMenuProc,
    0b00000000000000000000000000010000,
    enabled,
    "Outline",                          /*menu title*/
    {                                   /*menu items*/
        "Expand",           noicon, "E",   nomark, plain;
        "Expand To…",       noicon, nokey, nomark, plain;
        "Expand All",       noicon, nokey, nomark, plain;
        "Collapse",         noicon, nokey, nomark, plain;
        "-",                noicon, nokey, nomark, plain;
        /*the Label Style item has a submenu with menu ID mSubMenu*/
        "Label Style",      noicon, hierarchicalMenu, mSubMenu, plain;
        "-",                noicon, nokey, nomark, plain;
        "Move Left",        noicon, "L",   nomark, plain;
        "Move Right",       noicon, "R",   nomark, plain;
        "Move Up",          noicon, "U",   nomark, plain;
        "Move Down",        noicon, "D",   nomark, plain
    }
};
```

```
resource 'MENU' (mSubMenu , preload) {
    mSubMenu ,                          /*menu ID*/
    textMenuProc,
    0b00000000000000000000000001111111,
    enabled,
    "Label Style",                      /*menu title (ignored--defined */
                                        /* by parent menu item text)*/
    {                                   /*menu items*/
        "Alphabetic",       noicon, nokey, nomark, plain;
        "Bullet",           noicon, nokey, nomark, plain;
        "Chicago",          noicon, nokey, nomark, plain;
        "Harvard",          noicon, nokey, nomark, plain;
        "Legal",            noicon, nokey, nomark, plain;
        "Numeric",          noicon, nokey, nomark, plain;
        "Roman",            noicon, nokey, nomark, plain
    }
};
```

When you use `GetNewMBar` to read in menu descriptions and create a new menu list, `GetNewMBar` records the menu ID of any submenu in the menu record but does not read in the description of the submenu. To read a description of a submenu, use the `GetMenu` function. To actually insert a submenu into the current menu list, you must use the `InsertMenu` procedure.

When needed, your application can use the `GetMenu` function to read a description of the characteristics of a menu from a `'MENU'` resource. The `GetMenu` function creates a menu record for the menu, allocating space for the menu record in your application's heap. The `GetMenu` function creates the menu and menu items (and fills in the menu record) according to its `'MENU'` resource. The `GetMenu` function does not insert the menu into a menu list. When you're ready to add it to the current menu list, use the `InsertMenu` procedure.

Listing 3-8 uses the `GetMenu` function to read the description of a submenu and uses the `InsertMenu` procedure to insert the menu into the current menu list.

Listing 3-8   Creating a hierarchical menu

```
PROCEDURE MyMakeSubMenu (subMenuResID: Integer);
VAR
    subMenu: MenuHandle;
BEGIN
    subMenu := GetMenu(subMenuResID);
    InsertMenu(subMenu, -1);
END;
```

To insert a submenu into the current menu list using the `InsertMenu` procedure, specify –1 in the second parameter to insert the menu into the submenu portion of the menu list. As the user traverses menu items, if a menu item has a submenu the `MenuSelect` function looks in the submenu portion of the menu list for the submenu; it searches for a menu with a defined menu ID that matches the menu ID specified by the hierarchical menu item. If it finds a menu with a matching menu ID, it attaches the submenu to the menu item and allows the user to browse through the submenu.

Creating a Pop-Up Menu

In System 7, pop-up menus are implemented as controls. You define the menu items of a pop-up menu in the same way as in other menus (using a `'MENU'` resource), and you define specific features of the pop-up menu itself (such as the location of the pop-up menu) in a control that uses the standard pop-up control definition function. Pop-up menus provide the user with a simple way to select from among a list of choices without having to move up to the menu bar. They are particularly useful in a dialog box that requires the user to specify a number of settings or values. Figure 3-33 shows an example of a pop-up menu in a dialog box.

Figure 3-33   A pop-up menu in a dialog box
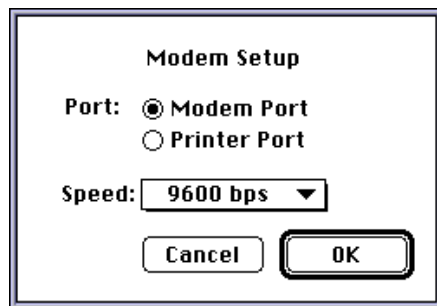
To create a pop-up menu, create a control that uses the pop-up control definition function, define the pop-up menu and its menu items, and associate the control with a window or dialog box. You can use Dialog Manager or Control Manager routines to display pop-up menus.

For example, if you define a modal dialog box that contains a pop-up control and use the Dialog Manager to display and help handle events in the dialog box, the Dialog Manager automatically uses the pop-up control definition function to draw the control and also to handle user interaction when the user presses the mouse button while the cursor is over a pop-up control.

If your application defines a control in one of your application's windows, you can use `TrackControl` and other Control Manager routines to handle the pop-up menu.

The pop-up control definition function draws a box around the pop-up box, draws the drop shadow, inserts the text into the pop-up box, draws the downward-pointing triangle, and draws the pop-up title. When a dialog box contains a control that uses the

pop-up control definition function and the user presses the mouse button while the cursor is in the pop-up control, the pop-up control definition function highlights the pop-up menu title, displays the pop-up menu, and handles all user interaction until the user releases the mouse button. When the user releases the mouse button, the pop-up control definition function closes the pop-up box, draws the user's choice in the pop-up box (or restores the previous item if the user did not make a new choice), stores the user's choice as the value of the control, and unhighlights the pop-up menu title. Your application can use the Control Manager function `GetControlValue` to get the value of the control and to determine the currently selected item in the pop-up menu.

To create a pop-up control, create a control and specify that the control uses the pop-up control definition function by specifying the `popupMenuProc` constant:

```
CONST popupMenuProc = 1008;        {pop-up menu control}
```

If you specify `popupMenuProc` (plus any appropriate variation code) as the `procID` field of the resource description of a control, when your application creates the control (by using the Dialog Manager or by using `GetNewControl`) the Control Manager creates the pop-up control, which includes the pop-up title and the pop-up box with a one-pixel drop shadow. The appearance of the pop-up title and the values in the menu are controlled by other values stored in the resource (or other parameters passed to `NewControl`). See the chapter "Control Manager" in this book for information on the `NewControl` function.

If your application does not use the standard pop-up control definition function, you can create your own control definition function to implement pop-up menus. In this case you can use the `PopUpMenuSelect` function to draw the pop-up menu and track the cursor within the menu. Your application is responsible for highlighting the title of the pop-up menu before calling `PopUpMenuSelect` and unhighlighting the title afterward (to duplicate the behavior of menu titles in the menu bar). Your application must also set the mark of the items in the pop-up menu as appropriate if you use the `PopUpMenuSelect` function.

For more information on creating controls, see the chapter "Control Manager" in this book. For listings that define the dialog box shown in Figure 3-33, see the chapter "Dialog Manager" in this book.

Changing the Appearance of Items in a Menu

You can change the appearance of an item in a menu using Menu Manager routines. For example, you can change the font style, text, or other characteristics of menu items. You can also enable or disable a menu item.

Most of the Menu Manager routines that get or set characteristics of a particular menu item require three parameters:

- a handle to the menu record of the menu containing the desired menu item
- the number of the menu item
- a variable that either specifies the data to set or identifies where to return information about that item

Enabling and Disabling Menu Items

Using the `EnableItem` and `DisableItem` procedures, you can enable and disable specific menu items or an entire menu. You pass as parameters to these two procedures a handle to the menu record that identifies the desired menu and either an item number that identifies the particular menu item to enable or disable or a value of 0 to indicate that the entire menu should be enabled or disabled.

Your application should always enable and disable any menu items as appropriate— according to the user's content—before calling `MenuSelect` or `MenuKey`. For example, you should enable the Paste command when the scrap contains data that the user can paste. (Listing 3-19 on page 3-74 shows code that adjusts an application's menus.)

When you disable or enable an entire menu, call `DrawMenuBar` to update the menu bar. The `DrawMenuBar` procedure draws the menus in the menu bar according to their current enabled state and as they are defined in the current menu list.

If you disable an entire menu, the Menu Manager dims the menu title at your application's next call to `DrawMenuBar` and dims all items in the menu when it displays the menu. If you enable an entire menu, the Menu Manager enables only the menu title and any items that you did not previously disable individually; the Menu Manager does not enable any item that your application previously disabled by calling `DisableItem` with that menu item's item number. For example, if all items in your application's Edit menu are enabled, you can disable the Cut and Copy commands individually using `DisableItem`. If you choose to disable the entire menu by passing 0 as the menu item parameter to `DisableItem`, the menu and all its items are disabled. If you then enable the entire menu by passing 0 as the menu item parameter to `EnableItem`, the menu and its items are enabled, except for the Cut and Copy commands, which remain disabled. In this case, to enable the Cut and Copy commands, you must enable each one individually using `EnableItem`.

You can use `DisableItem` to disable items that aren't appropriate at a given time. For example, you can disable the Cut and Copy commands when the user has not selected anything to cut or copy and disable the Paste command when the scrap is empty.

This code enables the File menu, disables the Cut and Copy commands in the Edit menu, and disables the application-defined menu Colors.

```
VAR
   menu: MenuHandle;

   menu := GetMenuHandle(mFile); {get a handle to the File menu}
   EnableItem(menu, 0);          {enable File menu and any items }
                                 { not individually disabled}
   DrawMenuBar;                  {update menu bar's appearance}

   menu := GetMenuHandle(mEdit); {get a handle to the Edit menu}
   DisableItem(menu, iCut);      {disable the Cut command}
   DisableItem(menu, iCopy);     {disable the Copy command}
```

```
menu := GetMenuHandle(mColors);{get a handle to Colors menu}
DisableItem(menu, 0);           {disable Colors menu & all }
                                { items in it}
DrawMenuBar;                    {update menu bar's appearance}
```

If you disable or enable an entire menu, call `DrawMenuBar` when you need to update the menu bar's appearance. If you do not need to update the menu bar immediately, you can use the `InvalMenuBar` procedure instead of `DrawMenuBar`, thus reducing flickering in the menu bar. Rather than drawing the menu bar twice as in the previous example, you can use `InvalMenuBar` instead of `DrawMenuBar`, causing the Event Manager to redraw the menu bar the next time it scans for update events. The `InvalMenuBar` procedure is available in System 7 and later. See page 3-114 for additional details on the `InvalMenuBar` procedure.

Changing the Text of an Item

You can get or set the text of a menu item using Menu Manager routines.

To get the text of a menu item, use the `GetMenuItemText` procedure. For example, you can use the `GetMenuItemText` procedure to get the text of a menu item that you added to a menu using `InsertResMenu` or `AppendResMenu`.

To set the text of a menu item, use the `SetMenuItemText` procedure. You can use the `SetMenuItemText` procedure as a convenient way to change the text of a menu command that allows the user to toggle between two states. For example, if your application has a menu command that allows the user to either show or hide the Clipboard window, depending on whether the window is currently showing, you can change the text of the menu item at the appropriate time using the `SetMenuItemText` procedure.

Listing 3-9 changes the text of a menu item from Hide Clipboard to Show Clipboard or vice versa, based on the state of an application-defined global variable (`gToggleState`) that holds the state information.

Listing 3-9   Changing the text of a menu item

```
PROCEDURE MyToggleHideShow;
VAR
   myMenu:     MenuHandle;
   item:       Integer;
   itemString: Str255;
BEGIN
   myMenu := GetMenuHandle(mEdit);
   item := iToggleHideShow;
   IF gToggleState = kShow THEN
   BEGIN
      GetIndString(itemString, kMyStrings, kShowClipboard);
      gToggleState := kHide;
   END
```

```
   ELSE
   BEGIN
      GetIndString(itemString, kMyStrings, kHideClipboard);
      gToggleState := kShow;
   END;
   SetMenuItemText(myMenu, item, itemString);
END;
```

Note that if you use the SetMenuItemText procedure, you should define the text of the menu item in a string resource or string list resource (for example, using an 'STR ' or 'STR#' resource). This makes your application easier to localize.

Changing the Font Style of Menu Items

You can change or get the font style of a menu item using the SetItemStyle or GetItemStyle procedure. To set the style of a menu item, specify a handle to the menu record of the menu containing the menu item whose style you want to set, specify the number of the menu item to set, and specify the desired style.

You specify the style using values from the set defined by the Style data type:

```
TYPE
      StyleItem = (bold, italic, underline, outline, shadow,
                      condense, extend);
      Style = SET OF StyleItem;
```

You can set the style of a menu item to zero, one, or more than one of the styles defined by the StyleItem data type. You can set the style of a menu item to the empty set to obtain the plain font style.

Listing 3-10 shows code that sets the style of menu items listed in an application's Style menu.

Listing 3-10   Setting the font style of menu items

```
VAR
   menu:       MenuHandle;
   itemStyle:  Style;

   menu := GetMenuHandle(mStyle); {get a handle to the Style menu}
   itemStyle := [italic];
   SetItemStyle(menu, iItalic, itemStyle);{set to italic style}
   itemStyle := [bold];
   SetItemStyle(menu, iBold, itemStyle);{set item to bold style}
   itemStyle := [bold, Italic];
   SetItemStyle(menu, iBoldItal, itemStyle);{bold & italic style}
   itemStyle := [];
   SetItemStyle(menu, iPlain, itemStyle);{set item to plain style}
```

To get the style of a menu item, you can use the GetItemStyle procedure.

Changing the Mark of Menu Items

You can change or get the mark of a menu item using the `SetItemMark` or
`GetItemMark` procedure. To set the mark of a menu item to a checkmark, you
can use either the `CheckItem` or the `SetItemMark` procedure.

To set the mark of a menu item, specify a handle to the menu record of the menu
containing the item whose mark you want to set, specify the number of the menu
item to set, and specify the mark to use as the marking character of the menu item.

You typically use checkmarks and dashes in menus that contain commands that set
attributes and that you have grouped in accumulating groups. For example, you use a
combination of checkmarks and dashes in the Style menu to indicate whether the
selection contains more than one style. Figure 3-8 on page 3-15 shows an example of
using checkmarks and dashes in a menu. "Groups of Menu Items" beginning on
page 3-14 gives guidelines for determining how to group your menu items.

You specify the mark of the menu item by passing a character as one of the parameters to
the `SetItemMark` procedure. You should use only the standard marking characters,
such as the checkmark, diamond, or dash, in your menu items; avoid using other marks
that might confuse the user. You can use the constants listed here to specify that the item
has no mark or to set the marking character to a checkmark or diamond:

```
CONST noMark      = 0;     {no marking character}
      checkMark   = $12;   {checkmark}
      diamondMark = $13;   {diamond symbol}
```

As another example of the use of marks in menus, Listing 3-11 shows code that sets the
mark of items in an application-defined Directory menu. It sets the marking character of
the menu item of the last directory accessed to a checkmark, sets the marking character
of the second-last directory accessed to the diamond mark, and removes the mark from
the third-last directory accessed.

Listing 3-11   Adding marks to and removing marks from menu items

```
VAR
   menu:       MenuHandle;
   itemMark:   Char;
                                 {get handle to Directory menu}
   menu := GetMenuHandle(mDirectory);

   itemMark := CHR(checkMark);
   SetItemMark(menu, gLastDir, itemMark); {set mark to checkmark}

   itemMark := CHR(diamondMark);
   SetItemMark(menu, gOldLastDir, itemMark); {set mark to diamond}

   itemMark := CHR(noMark);
   SetItemMark(menu, gSecondLastDir, itemMark);{remove any mark}
```

You can also set the mark of a menu item to a checkmark using the `CheckItem`
procedure:

```
VAR
   menu:         MenuHandle;
                                 {get handle to Directory menu}
   menu := GetMenuHandle(mDirectory);
   CheckItem(menu, gLastDir, TRUE);         {set to checkmark}
   CheckItem(menu, gSecondLastDir, FALSE);{remove checkmark or }
                                      { any other mark}
```

Changing the Icon or Script Code of Menu Items

You can change or get the icon of a menu item using the `SetItemIcon` or
`GetItemIcon` procedure. You can also use these procedures to get or set the
script code of a menu item's text.

To set the script code of a menu item using the `SetItemIcon` procedure, you need to

■ specify a handle to the menu record of the menu containing the item whose script
  code you want to set

■ specify the number of the menu item to set

■ specify the script code

To set a menu item's script code, you must also define the keyboard equivalent field of
the item to $1C. If an item contains $1C in its keyboard equivalent field and a script code
in its icon field, the Menu Manager draws the item in the script identified by the script
code value if the corresponding script system is installed.

To set the icon of a menu item using the `SetItemIcon` procedure, you need to

■ specify a handle to the menu record of the menu containing the item whose icon you
  want to set

■ specify the number of the menu item to set

■ specify the icon number (the Menu Manager uses the icon number to generate the
  resource ID of the icon)

The icon number that you specify to `SetItemIcon` must be a value from 1 through 255
for color icons or icons, from 1 through 254 for small icons and reduced icons, or 0 to
specify that the item doesn't have an icon. The Menu Manager adds 256 to the number
you specify and uses this calculated number as the icon's resource ID. For example, if
you specify the icon number as 5, the Menu Manager uses the Resource Manager to find
the icon with resource ID 261. The Menu Manager first looks for an icon resource of type
`'cicn'`; if it can't find one with the calculated resource ID number (or if the computer
doesn't have Color QuickDraw), it looks for a resource of type `'SICN'` if the keyboard
equivalent field contains $1E; otherwise, it looks for an `'ICON'` resource.

Use either an `'ICON'` or `'SICN'` resource if you want to provide only a black-and-white
icon. In addition, provide a `'cicn'` resource if you want the Menu Manager to use a
color icon when Color QuickDraw is available. Figure 3-34 shows examples of icons in a
menu item generated from icon resources: an `'SICN'` resource, an `'ICON'` resource, an
`'ICON'` resource reduced to fit in a 16-by-16 bit rectangle, and a `'cicn'` resource.

Menu Manager

Figure 3-34   Icons in menu items
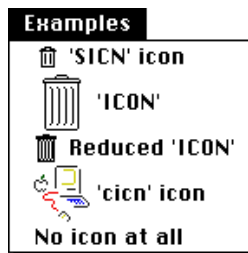
The Menu Manager automatically fits the icon in the menu item according to your specifications. If the Menu Manager uses a 'cicn' resource, it automatically enlarges the enclosing rectangle of the menu item according to the rectangle specified in the 'cicn' resource. If the Menu Manager uses an 'ICON' resource and the item specifies the nokey constant as the keyboard equivalent, the Menu Manager enlarges the rectangle of the menu item to fit the 32-by-32 bit 'ICON' resource. You can request that the Menu Manager reduce an 'ICON' resource to the size of a 16-by-16 bit small icon by specifying a value of $1D as the item's keyboard equivalent. To request that the Menu Manager use an 'SICN' resource instead of an 'ICON' resource, specify a value of $1E as the item's keyboard equivalent.

This code sets the icon of a menu item to a specified icon.

```
VAR
    menu:       MenuHandle;
    itemIcon:   Byte;

    itemIcon := 5;
    menu := GetMenuHandle(mWeather);
    {set the icon for this item in the Weather menu}
    SetItemIcon(menu, iBeachWeather, itemIcon);
```

Listing 3-12 shows the Rez description of three menu items, each of which contains icons. The first menu item has an icon with resource ID 261 (5 plus 256) and is defined by a resource type of either 'cicn' or 'ICON'. The second menu item has an icon with resource ID 262 (6 plus 256) and is identified by either a 'cicn' resource or an 'ICON' resource; however, in this case, the value of $1D requests the Menu Manager to reduce the 'ICON' resource to a small icon. The third menu item has an icon with resource ID 263 (7 plus 256) and is defined by either a 'cicn' resource or an 'SICN' resource.

Listing 3-12   Specifying icons for menu items

```
#define mWeather 138
resource 'MENU' (mWeather, preload) {
      mWeather,
      textMenuProc,
```

```
    0b0000000000000000001011101100111,
    enabled, "Weather",
    {
        "Beach Weather",  /*item has icon or color icon */
                          /* with icon number 5*/
           5, nokey, nomark, plain;
        "Ski Weather",    /*item has reduced icon or color */
                          /* icon with icon number 6*/
           6, $1D, nomark, plain;
        "Kite-Flying Weather",/*item has small icon or */
                               /* color icon with icon number 7*/
           7, $1E, nomark, plain
    }
};
```

See the chapter "Finder Interface" in this book for details on how to create icons.

Adding Items to a Menu

Usually you define a menu and all its items in a `'MENU'` resource. Occasionally you might need to add items to a menu after you've created it. After creating a menu (using `NewMenu`, `GetMenu`, or `GetNewMBar`), you can add items to it using the `AppendMenu`, `InsertMenuItem`, `AppendResMenu`, or `InsertResMenu` procedure.

You can use `AppendResMenu` or `InsertResMenu` to add items that consist of resource names to a menu. For example, you can use the `AppendResMenu` procedure to add fonts to your application's Font menu or to add all of the desktop items from the Apple Menu Items folder to your application's Apple menu. These are common instances when you'll need to add items not already defined in a `'MENU'` resource to a menu. See "Adding Fonts to a Menu" on page 3-69 and "Adding Items to the Apple Menu" on page 3-68 for information on adding names of resources to menus.

If you add items to your application's Help menu, you'll need to use `AppendMenu` or `InsertMenuItem` to add the additional items. This section discusses how to add items using the `AppendMenu` and `InsertMenuItem` procedures, and "Adding Items to the Help Menu" on page 3-67 shows a specific example of adding items to the Help menu.

If you need to add items other than the names of resources to a previously created menu, you can use the `AppendMenu` or `InsertMenuItem` procedure. You can use `AppendMenu` to add items to the end of a menu; note that you can add items to only the end of the menu when using `AppendMenu`. Use `InsertMenuItem` to add items after any given item in a menu. When you add items to a menu using `AppendMenu` or `InsertMenuItem`, you can specify the same characteristics for menu items that are available to you when defining `'MENU'` resources.

You specify a handle to the menu record of the menu to which you want to add the item or items, and you specify a string describing the items to add as parameters to the `AppendMenu` or `InsertMenuItem` procedure. The string you pass to these procedures should consist of the text and any characteristics of the menu items. You can specify a hyphen as the menu item text to create a divider line. You can also use various metacharacters in the text string to separate menu items and to specify certain characteristics of the menu items. The metacharacters aren't displayed in the menu.

Here is a list of the metacharacters you can use in the text string that you specify to the `AppendMenu` or `InsertMenuItem` procedure:

Metacharacter	Description
; or Return	Separates menu items.
^	When followed by an icon number, defines the icon for the item. If the keyboard equivalent field contains $1C, this number is interpreted as a script code.
!	When followed by a character, defines the mark for the item. If the keyboard equivalent field contains $1B, this value is interpreted as the menu ID of a submenu of this menu item.
<	When followed by one or more of the characters B, I, U, O, and S, defines the character style of the item to Bold, Italic, Underline, Outline, or Shadow, respectively.
/	When followed by a character, defines the keyboard equivalent for the item. When followed by $1B, specifies that this menu item has a submenu. To specify that the menu item has a script code, small icon, or reduced icon, use the `SetItemCmd` procedure to set the keyboard equivalent field to $1C, $1D, or $1E, respectively.
(	Defines the menu item as disabled.

You can specify any, all, or none of these metacharacters in the text string to define the characteristics of a menu item. Note that the metacharacters that you specify aren't displayed in the menu item. (To use any of these metacharacters in the text of a menu item, first use `AppendMenu` or `InsertMenuItem`, specifying at least one character as the item's text. Then use the `SetMenuItemText` procedure to set the item's text to the desired string.)

Note

If you add menu items using the `AppendMenu` or `InsertMenuItem` procedure, you should define the text and any marks or keyboard equivalents in resources for easier localization. ◆

Listing 3-13 shows a string list (`'STR#'`) resource that stores the text of the menu items used in the next examples.

Menu Manager

Listing 3-13   Rez input for text of menu items

```
resource 'STR#' (300, "Text for appended menu items") {
    {
    /*[1]*/
    "Just Text";
    /*[2]*/
    "Pick a Color…";
    /*[3]*/
    "(^2!=Everything<B/E";
    }
);
```

Here's code that uses the AppendMenu procedure to append a menu item with no specific characteristics other than its text to the menu identified by the menu handle in the myMenu variable. The text for the menu item is "Just Text" as stored in the 'STR#' resource with resource ID 300.

```
VAR
    myMenu:     MenuHandle;
    itemString: Str255;

    myMenu := GetMenuHandle(mLibrary);
    GetIndString(itemString, 300, 1);
    AppendMenu(myMenu, itemString);
```

To insert an item after a given menu item, use the InsertMenuItem procedure. For example, this code inserts the menu item Pick a Color after the menu item with the item number specified by the iRed constant. The text for the new menu item consists of the string "Pick a Color…" as stored in the 'STR#' resource with resource ID 300.

```
VAR
    myMenu:     MenuHandle;
    itemString: Str255;

    myMenu := GetMenuHandle(mColors);
    GetIndString(itemString, 300, 2);
    InsertMenuItem(myMenu, itemString, iRed);
```

If you do not explicitly specify a value for an item characteristic in the text string that you pass to AppendMenu or InsertMenuItem, the procedure assigns the default value for that characteristic. The Menu Manager defines the default item characteristics as no icon, no marking character, no keyboard equivalent, and plain text style. AppendMenu and InsertMenuItem enable the added menu items unless you specify otherwise.

This code appends a menu item with the text "Everything" to the menu identified by the menu handle in the myMenu variable. The text and other characteristics of this menu item are stored in the 'STR#' resource shown in Listing 3-13. It also specifies that this

menu item is disabled, has an icon with resource ID 258 (2 + 256), and has the "="
character as a marking character; the style of the text is Bold; and the menu item has a
keyboard equivalent of Command-E.

```
VAR
    myMenu:     MenuHandle;
    itemString: Str255;

    myMenu := GetMenuHandle(mLibrary);
    GetIndString(itemString, 300, 3);
    AppendMenu(myMenu, itemString);
```

This code appends multiple items to the Edit menu using `AppendMenu`:

```
VAR
    myMenu:     MenuHandle;

    myMenu := GetMenuHandle(mEdit);
    AppendMenu(myMenu, 'Undo/Z;-;Cut/X;Copy/C;Paste/V');
```

The `InsertMenuItem` procedure differs from `AppendMenu` in how it handles the given
text string when the text string specifies multiple items. The `InsertMenuItem`
procedure inserts the items in the reverse of their order in the text string. For example,
this code inserts menu items into the Edit menu using `InsertMenuItem` and is
equivalent to the previous example:

```
VAR
    myMenu:     MenuHandle;

    myMenu := GetMenuHandle(mEdit);
    InsertMenuItem(myMenu, 'Paste/V';Copy/C;Cut/X;-;Undo/Z',0);
```

Once you've added a menu item to a menu, you can change any of its characteristics
using Menu Manager routines, as described in "Changing the Appearance of Items in a
Menu" on page 3-57.

Adding Items to the Help Menu

You add items to the Help menu by using the `HMGetHelpMenuHandle` function and
either the `AppendMenu` or `InsertMenuItem` procedure.

The `HMGetHelpMenuHandle` function returns a copy of the handle to the menu record
of your application's Help menu. Do not use the `GetMenuHandle` function to get a
handle to the Help menu, because `GetMenuHandle` returns a handle to the global Help
menu, not the Help menu that is specific to your application. Once you have a handle to
the Help menu that is specific to your application, you can add items to it using the
`AppendMenu` procedure or other Menu Manager routines. For example, Listing 3-14
adds the menu item displayed in Figure 3-19 on page 3-30.

Listing 3-14   Adding an item to the Help menu

```
PROCEDURE MyAddHelpItem;
VAR
    myMenu:     MenuHandle;
    myErr:      OSErr;
    itemString: Str255;
BEGIN
    myErr := HMGetHelpMenuHandle(myMenu);
    IF myErr = noErr THEN
        IF myMenu <> NIL THEN
        BEGIN
            {get the string (with index kSurfHelp) from the 'STR#' }
            { resource with resource ID kMyStrings}
            GetIndString(itemString, kMyStrings, kSurfHelp);
            AppendMenu(myMenu, itemString);
        END;
END;
```

When you add items to the Help menu, the Help Manager automatically adds a divider between the end of the standard Help menu items and your items.

Be sure to use an 'hmnu' resource and specify the kHMHelpMenuID constant as the resource ID to provide help balloons for items you've added to the Help menu. (The Help Manager automatically creates the help balloons for the Help menu title and the standard Help menu items.) See the chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox* for specific information on the 'hmnu' resource.

The Help Manager automatically processes the event when a user chooses any of the standard menu items in the Help menu. The Help Manager automatically enables and disables help when the user chooses Show Balloons or Hide Balloons from the Help menu. The setting of Balloon Help is global and affects all applications. See "Handling the Help Menu" on page 3-81 for information on responding to the user when the user chooses one of your appended items.

Adding Items to the Apple Menu

To insert the items contained in the Apple Menu Items folder into your application's Apple menu, use the AppendResMenu or InsertResMenu procedure and specify 'DRVR' as the resource type. Doing so causes this procedure to automatically add all items in the Apple Menu Items folder to the specified menu.

The user can place any desktop object in the Apple Menu Items folder. When the user places an item in this folder, the system software automatically adds it to the list of items in the Apple menu of all open applications.

After inserting the Apple menu into your application's menu bar (by using `GetNewMBar`
or `GetMenu` and `InsertMenu`), your application can add items to it. Listing 3-15 shows
code that uses `GetMenuHandle` to get a handle to the application's Apple menu. The
code then uses the `AppendResMenu` procedure, specifying that `AppendResMenu` should
add all desktop items in the Apple Menu Items folder to the application's Apple menu.

Listing 3-15 Adding menu items to the Apple menu

```
VAR
   myMenu:            MenuHandle;

   myMenu := GetMenuHandle(mApple);
   IF myMenu <> NIL THEN
      AppendResMenu(myMenu, 'DRVR');{add desktop items in the }
                                    { Apple Menu Items folder }
                                    { to the Apple menu}
```

Listing 3-16 on page 3-70 shows a complete sample that sets up an application's menu
bar, adds items to the Apple menu, adds items to the Font menu, and then updates the
appearance of the menu bar.

Adding Fonts to a Menu

If your application provides a Font menu, you typically include the description of the
menu in a `'MENU'` resource, include a description of your menu bar in an `'MBAR'`
resource, and use `GetNewMBar` to create your menu bar and all menus in the menu bar.
Once you've created the menu, you can use `AppendResMenu` to add the names of all
font resources in the Fonts folder of the System Folder (or in system software versions
earlier than 7.1, in the System file) as menu items in your application's Font menu. (You
can also use `InsertResMenu` to insert the fonts into your menu.)

Listing 3-16 on the next page shows how to add names of font resources in the Fonts
folder to an application's Font menu. The `AppendResMenu` procedure adds all resources
of the specified type to a given menu. If you specify the resource type `'FONT'` or
`'FOND'`, the Menu Manager adds all resources of type `'FOND'` and `'FONT'` to the
menu. (`'NFNT'` and `'sfnt'` resources are specified through `'FOND'` resources.)

The `AppendResMenu` and `InsertResMenu` procedures perform special processing for
any font resources they find that have font numbers greater than $4000. The Menu
Manager automatically sets the keyboard equivalent field of the menu item to $1C and
stores the script code in the icon field of the menu item for any such `'FOND'` resource.
The Menu Manager displays a font name in its corresponding script if the script system
for that font is installed.

3

Menu Manager

Listing 3-16   Adding font names to a menu

```
PROCEDURE MyMakeAllMenus;
VAR
   menu:                 MenuHandle;
   menuBar:              Handle;
 BEGIN
                         {read menus in & create new menu list}
   menuBar := GetNewMBar(rMenuBar);
   IF menuBar = NIL THEN
      EXIT(MyMakeAllMenus);
   SetMenuBar(menuBar); {insert menus into the current menu list}
   DisposHandle(menuBar);
   myMenu := GetMenuHandle(mApple);
   IF myMenu <> NIL THEN                 {add desktop items in }
      AppendResMenu(myMenu, 'DRVR');     { Apple Menu Items }
                                         { folder to Apple menu}
   myMenu := GetMenuHandle(mFont);
   IF myMenu <> NIL THEN
      AppendResMenu(myMenu, 'FONT');     {add font names to the }
         { Font menu--this adds all bitmapped and TrueType fonts }
         { in the Fonts folder to the Font menu}
   MyAddHelpItem;       {add app-specific item to Help menu}
   MyAdjustMenus;       {adjust menu items}
   DrawMenuBar;         {draw the menu bar}
 END;
```

Your application should indicate the current font to the user by placing the appropriate mark next to the text of the menu item that lists the font name. ("Changing the Mark of Menu Items" on page 3-61 explains how to add marks to and remove marks from menu items; Figure 3-13 on page 3-26 and Figure 3-14 on page 3-27 show examples of typical Font menus.)

If your application allows the user to change the font style or font size of text, you should provide separate Size and Style menus. See "Handling a Size Menu" beginning on page 3-82 for information on providing a Size menu in your application.

Handling User Choice of a Menu Command

If the user presses the mouse button while the cursor is in the menu bar, your application should first adjust its menus (enable or disable menu items and add marks to or remove marks from any items as appropriate to the user's context) and then call the `MenuSelect` function to allow the user to choose a menu command. The `MenuSelect` function handles all user interaction until the user releases the mouse button and returns a value as its function result that indicates which (if any) menu item the user chose.

For a command with a keyboard equivalent, your application should allow the user to choose the command by pressing the keys that correspond to the keyboard equivalent of that menu command. If the user presses the Command key and another key, your application should adjust its menus and then call the MenuKey function to map this combination to a keyboard equivalent. The MenuKey function returns as its function result a value that indicates the corresponding menu and menu item of the keyboard equivalent.

When the user chooses a menu command, your application should perform the action associated with that command. The MenuSelect and MenuKey functions highlight the menu title of the menu containing the chosen menu command. After your application performs any operation associated with the menu command chosen by the user, your application should unhighlight the menu title by using the HiliteMenu procedure.

However, if in response to a menu command your application displays a window that contains editable text (such as a modal dialog box), you should unhighlight the menu title immediately so that the user can access the Edit menu or other appropriate menus. In other words, any time the user can use a menu, make sure that the menu title is not highlighted.

When the user chooses a menu command that involves an operation that takes a long time, display the animated wristwatch cursor or display a status dialog box to give the user feedback that the operation is in progress.

If you want the users of your application to be able to record their actions (such as menu commands, text input, or any sequence of actions) for later playback, your application should send itself Apple events whenever a user performs a significant action. To do this for menu commands, your application typically sends itself an Apple event to perform the action associated with the chosen menu command. For example, when a user chooses the New command from the File menu, your application can choose to send itself a Create Element event. Your application then creates the new document in response to this event. For information on sending Apple events in response to menu commands, see *Inside Macintosh: Interapplication Communication*.

The next sections show how your application can

■ determine if the user pressed the mouse button while the cursor was in the menu bar

■ adjust its menus—enabling and disabling commands according to the current state of the document—before displaying menus or before responding to the user's choice of a keyboard equivalent of a command

■ determine if the user chose the keyboard equivalent of a menu command

■ respond to the user when the user chooses a menu command

The next sections also show how your application should respond when the user chooses an item from the Apple or Help menu.

Handling Mouse-Down Events in the Menu Bar

You can determine when the user has pressed the mouse button while the cursor is in the menu bar by examining the event record for a mouse-down event. You can use the Window Manager function `FindWindow` to map the mouse location at the time of the mouse-down event to a corresponding area of the screen. If the cursor was in the menu bar, your application should call the `MenuSelect` function, allowing the user to choose a menu command.

Listing 3-17 shows an application-defined procedure, `DoEvent`, that determines whether a mouse-down event occurred and, if so, calls another application-defined procedure to handle the mouse-down event. (For a complete discussion of how to handle events, see the "Event Manager" chapter in this book.)

Listing 3-17   Determining whether a mouse-down event occurred

```
PROCEDURE DoEvent (event: EventRecord);
BEGIN
   CASE event.what OF
      mouseDown:                      {handle mouse-down event}
         DoMouseDown(event);
      {handle other events appropriately}
   END; {of CASE}
END;
```

Listing 3-18 shows an application-defined procedure, `DoMouseDown`, that handles mouse-down events. The `DoMouseDown` procedure determines where the cursor was when the mouse button was pressed and then responds appropriately.

Listing 3-18   Determining when the cursor is in the menu bar

```
PROCEDURE DoMouseDown (event: EventRecord);
VAR
   part:       Integer;
   thisWindow: WindowPtr;
BEGIN
   part := FindWindow(event.where, thisWindow);
   CASE part OF
      inMenuBar:{mouse down in menu bar, respond appropriately}
         BEGIN
            {adjust marks and enabled state of menu items}
            MyAdjustMenus;
            {let user choose a menu command if desired}
            DoMenuCommand(MenuSelect(event.where));
         END;
```

```
      {handle other mouse-down events appropriately}
   END; {of CASE}
END;
```

You can use the `FindWindow` function to map the mouse location at the time the user pressed the mouse button to general areas of the screen. If the mouse location is in the menu bar, the `FindWindow` function returns the constant `inMenuBar`. In Listing 3-18, if the mouse location associated with the mouse-down event is in the menu bar, the `DoMouseDown` procedure first calls another application-defined procedure, `MyAdjustMenus`, to adjust the menus. Listing 3-19 shows the `MyAdjustMenus` procedure.

The `DoMouseDown` procedure then calls an application-defined procedure, `DoMenuCommand`. The `DoMouseDown` procedure passes as a parameter to the `DoMenuCommand` procedure the value returned from the `MenuSelect` function.

The `MenuSelect` function displays menus and handles all user interaction until the user releases the mouse button. The `MenuSelect` function returns a long integer indicating whether the user chose a menu command, and if so, it indicates which menu and which command the user chose.

Listing 3-24 on page 3-79 shows the `DoMenuCommand` procedure.

Adjusting the Menus of an Application

Your application should always adjust its menus before calling `MenuSelect` or `MenuKey`. For example, you should enable and disable any menu items as necessary and add checkmarks or dashes to items that are attributes. When you adjust your application's menus, you should enable and disable menu items according to the type of window that is in the front. For example, when a document window is the frontmost window, you should enable items as appropriate for that document window. When a modeless dialog box or modal dialog box is the frontmost window, enable those items as appropriate to that particular dialog box. Listing 3-19 shows an application-defined routine, `MyAdjustMenus`, that adjusts the menus of the SurfWriter application appropriately.

The `MyAdjustMenus` procedure first determines what kind of window is in front and then adjusts the application's menus appropriately. The application-defined `MyGetWindowType` procedure returns a value that indicates whether the window is a document window, a dialog window, or a window belonging to a desk accessory. It also returns the constant `kNil` if there isn't a front window. (See the chapter "Window Manager" in this book for a listing of the `MyGetWindowType` procedure.) The `MyAdjustMenus` procedure calls other application-defined routines to adjust the menus as appropriate for the given window type.

Listing 3-19   Adjusting an application's menus

```
PROCEDURE MyAdjustMenus;
VAR
   window:              WindowPtr;
   windowType:          Integer;
BEGIN
   window := FrontWindow;
   windowType := MyGetWindowType(window);
   CASE windowType OF

   kMyDocWindow:
   BEGIN {document window is in front, adjust items
appropriately}
      MyAdjustFileMenuForDocWindow;
      MyAdjustEditMenuForDocWindow;
      {adjust other menus as needed}
   END;  {of adjusting menus for a document window}

   kMyDialogWindow:
      {adjust menus accordingly for any dialog box}
      MyAdjustMenusForDialogs;

   kDAWindow:{adjust menus accordingly for a DA window}
      MyAdjustMenusForDA;
   kNil:{adjust menus accordingly when there isn't a front
window}
      MyAdjustMenusNoWindows;
   END; {of CASE}
   DrawMenuBar;
END;
```

Listing 3-20 shows the application-defined procedure
`MyAdjustFileMenuForDocWindow`. This procedure enables and disables the File
menu for the application's document window, according to the state of the document.
For example, this application always allows the user to create a new document or open
a file, so the code enables the New and Open menu items. The code also enables the
Close, Save As, Page Setup, Print, and Quit menu items. If the user has modified the
file since last saving it, the code enables the Save command; otherwise, it disables the
Save command.

Listing 3-20   Adjusting the File menu for a document window

```
PROCEDURE MyAdjustFileMenuForDocWindow;
VAR
   window:     WindowPtr;
```

Menu Manager

```
   menu:        MenuHandle;
   myData:      MyDocRecHnd;
BEGIN
   window := FrontWindow;
   menu := GetMenuHandle(mFile); {get a handle to the File menu}
   IF menu = NIL THEN            {add your own error handling}
      EXIT (MyAdjustFileMenuForDocWindow);
   EnableItem(menu, iNew);
   EnableItem(menu, iOpen);
   EnableItem(menu, iClose);
   myData := MyDocRecHnd(GetWRefCon(window));
   IF myData^^.windowDirty THEN
      EnableItem(menu, iSave)
   ELSE
      DisableItem(menu, iSave);
   EnableItem(menu, iSaveAs);
   EnableItem(menu, iPageSetup);
   EnableItem(menu, iPrint);
   EnableItem(menu, iQuit);
END;
```

Listing 3-21 shows the application-defined `MyAdjustEditMenuForDocWindow`
procedure.

Listing 3-21   Adjusting the Edit menu for a document window

```
PROCEDURE MyAdjustEditMenuForDocWindow;
VAR
   window:             WindowPtr;
   menu:               MenuHandle;
   selection, undo:    Boolean;
   isSubscriber:       Boolean;
   undoText:           Str255;
   offset:             LongInt;
BEGIN
   window := FrontWindow;
   menu := GetMenuHandle(mEdit); {get a handle to the Edit menu}
   IF menu = NIL THEN            {add your own error handling}
      EXIT (MyAdjustEditMenuForDocWindow);
   undo := MyIsLastActionUndoable(undoText);
   IF undo THEN   {if action can be undone}
   BEGIN
```

```
        SetMenuItemText(menu, iUndo, undoText);
        EnableItem(menu, iUndo);
     END

  ELSE             {if action can't be undone}
  BEGIN
     SetMenuItemText(menu, iUndo, gCantUndo);
     DisableItem(menu, iUndo);
  END;
  selection := MySelection(window);
  IF selection THEN
  BEGIN    {enable editing items if there's a selection}
     EnableItem(menu, iCut);
     EnableItem(menu, iCopy);
     EnableItem(menu, iCreatePublisher);
  END
  ELSE
  BEGIN    {disable editing items if there isn't a selection}
     DisableItem(menu, iCut);
     DisableItem(menu, iCopy);
     DisableItem(menu, iCreatePublisher);
  END;
  IF GetScrap(NIL, 'TEXT', offset) > 0 THEN
     EnableItem(menu, iPaste)   {enable if something to paste}
  ELSE
     DisableItem(menu, iPaste); {disable if nothing to paste}
  EnableItem(menu, iSelectAll);
  EnableItem(menu, iSubscribeTo);
  IF MySelectionContainsSubscriberOrPublisher(isSubriber) THEN
  BEGIN {selection contains a single subscriber or publisher}
     IF isSubscriber THEN {selection contains a subscriber}
        SetMenuItemText(menu, iPubSubOptions, gSubOptText)
     ELSE                 {selection contains a publisher}
        SetMenuItemText(menu, iPubSubOptions, gPubOptText);
     EnableItem(menu, iPubSubOptions);
  END
  ELSE  {selection contains either no subscribers or publishers }
        { or contains at least one subscriber and one publisher}
     DisableItem(menu, iPubSubOptions);
  IF (gPubCount > 0) OR (gSubCount > 0) THEN
     EnableItem(menu, iShowHideBorders)
  ELSE
     DisableItem(menu, iShowHideBorders);
END;
```

The procedure in Listing 3-21 adjusts the items in the Edit menu as appropriate for a document window of the application. The code enables the Undo command if the application can undo the last command, enables the Cut and Copy commands if there's a selection that can be cut or copied, enables the Paste command if there's text data in the scrap, and enables the menu items relating to publishers and subscribers appropriately, according to whether the current selection contains a publisher or subscriber. The application-defined `MySelectionContainsSubscriberOrPublisher` function returns `TRUE` if the current selection contains a single subscriber or a single publisher and returns `FALSE` otherwise. If the `MySelectionContainsSubscriberOrPublisher` function returns `TRUE`, the code sets the text for the Publisher Options (or Subscriber Options) command and enables the menu item. If the function returns `FALSE`, the code disables the Publisher Options (or Subscriber Options) command.

Determining if the User Chose a Keyboard Equivalent

Keyboard equivalents of commands allow the user to invoke a menu command from the keyboard. You can determine if the user chose the keyboard equivalent of a menu command by examining the event record for a key-down event. If the user pressed the Command key in combination with another 1-byte character, you can determine if this combination maps to a Command-key equivalent by using the `MenuKey` function.

If your application supports keyboard equivalents that use other modifier keys in addition to the Command key, your application should examine the `modifiers` field and take any appropriate action; depending on the modifier keys you use, your application may or may not be able to use `MenuKey` to map the key to the menu command.

Listing 3-22 shows an application-defined procedure, `DoEvent`, that determines whether a key-down event occurred and, if so, calls an application-defined routine to handle the key-down event.

Listing 3-22   Determining when a key is pressed

```
PROCEDURE DoEvent (event: EventRecord);
BEGIN
   CASE event.what OF
      keyDown, autoKey:        {handle keyboard events}
         DoKeyDown(event);
      {handle other events appropriately}
   END; {of CASE}
END;
```

If your application determines that the user pressed a key, you need to determine whether the user chose the keyboard equivalent of a menu command. You can do this by examining the `modifiers` field of the event record describing the key-down event. If the Command key was also pressed, then your application should call the `MenuKey` function. The `MenuKey` function scans the current menu list for a menu item that has a

matching keyboard equivalent and returns the menu and menu item, if any. Although you should not define the same keyboard equivalent for more than one command, the MenuKey function scans the menus from right to left, scanning the items from top to bottom, and returns the first matching keyboard equivalent that it finds.

If your application uses other keyboard equivalents in addition to Command-key equivalents, you can examine the state of the modifier keys and use the Event Manager function KeyTranslate, if necessary, to help map the keyboard equivalent to a particular menu item. See the discussion of 'KCHR' resources in *Inside Macintosh: Text* for information on how various keyboard combinations map to specific character codes.

Listing 3-23 shows an application's DoKeyDown procedure that handles key-down events and determines if a keyboard equivalent was pressed.

Listing 3-23   Checking a key-down event for a keyboard equivalent

```
PROCEDURE DoKeyDown (event: EventRecord);
VAR
   key:          Char;
BEGIN
   key := CHR(BAnd(event.message, charCodeMask));
   IF BAnd(event.modifiers, cmdKey) <> 0 THEN
   BEGIN         {Command key down}
      IF event.what = keyDown THEN
      BEGIN                    {first enable/disable/check }
         MyAdjustMenus;    { menu items properly}
         DoMenuCommand(MenuKey(key));{handle the menu command}
      END;
   END
   ELSE
      MyHandleKeyDown(event);
END;
```

Listing 3-23 extracts the pressed key from the message field of the event record and then examines the modifiers field to determine if the Command key was also pressed. If so, the application first adjusts its menus and then calls an application-defined procedure, DoMenuCommand. The DoKeyDown procedure passes as a parameter to the DoMenuCommand procedure the value returned from the MenuKey function.

Listing 3-24 shows the DoMenuCommand procedure.

Responding When the User Chooses a Menu Item

Your application can use the MenuSelect function to determine when the user chooses a menu command, and your application can use the MenuKey function to determine when the user presses the keyboard equivalent for a menu command. Both MenuSelect

and `MenuKey` return a long integer value that indicates which menu and menu item the user chose.

The `MenuSelect` and `MenuKey` functions return the menu ID of the menu in the high word and the menu item number in the low word of their function result. If the user did not choose a menu command or if the user pressed a keyboard combination that does not map to any keyboard equivalent in your application's menus, the functions return 0 in the high word and the value of the low word is undefined. The `MenuSelect` function also returns 0 in the high word when the user selects an item in the Application or Keyboard menu. The `MenuSelect` function (and `MenuKey` function, if the command has a keyboard equivalent) returns the `kHMHelpMenuID` constant in the high word and the menu item in the low word when the user selects an item that your application appended to the Help menu.

Listing 3-24 shows an application-defined procedure, `DoMenuCommand`. This procedure takes the appropriate action based on which menu command the user chose.

The `DoMenuCommand` procedure is called by the application after the application determines that either the user pressed the mouse button while the cursor was in the menu bar (in which case the application calls `MenuSelect` to allow the user to choose a command) or the user pressed the Command key and another key (in which case the application calls the `MenuKey` function). In either case, the application passes the function result returned by `MenuSelect` or `MenuKey` as a parameter to the `DoMenuCommand` procedure.

Listing 3-24   Responding to the user's choice of a menu command

```
PROCEDURE DoMenuCommand (menuResult: LongInt);
VAR
   menuID, menuItem: Integer;
BEGIN
   menuID := HiWord(menuResult);    {get menu ID of menu}
   menuItem := LoWord(menuResult);  {get menu item number}
   CASE menuID OF
      mApple:
         MyHandleAppleCommand(menuItem);
      mFile:
         MyHandleFileCommand(menuItem);
      mEdit:
         MyHandleEditCommand(menuItem);
      mFont:
         MyHandleFontCommand(menuItem);
      mSize:
         MyHandleSizeCommand(menuItem);
      kHMHelpMenuID:
         MyHandleHelpCommand(menuItem);
      mOutline:
```

```
        MyHandleOutlineCommand(menuItem);
     mSubMenu: {user chose item from submenu}
        MyHandleSubLabelStyleCommand(menuItem);
  END;  {end of CASE menuID}
  HiliteMenu(0); {unhighlight what MenuSelect or MenuKey hilited}
END;
```

The DoMenuCommand procedure calls other application-defined routines to perform the requested action. After performing the action associated with the chosen menu item, your application should use the HiliteMenu procedure to unhighlight the menu title to indicate that the requested action is complete.

Handling the Apple Menu

When the user chooses an item from the Apple menu, the MenuSelect function returns the menu ID of your application's Apple menu in the high word and returns the chosen menu item in the low word of its function result.

If your application provides an About command as the first menu item in the Apple menu and the user chose this item, you should display your application's About box. Otherwise your application should use the GetMenuItemText procedure to get the menu item text and then call the OpenDeskAcc function, passing the text of the chosen menu item as a parameter.

Listing 3-25 shows an application-defined procedure, MyHandleAppleCommand, that the application calls in response to the user's choice of an item from the Apple menu.

Listing 3-25   Responding to the user's choice of an item from the Apple menu

```
PROCEDURE MyHandleAppleCommand (menuItem: Integer);
VAR
   itemName: Str255;
   daRefNum: Integer;
BEGIN
   CASE menuItem OF
      iAbout:             {bring up alert for About}
         DisplayMyAboutBox;
      OTHERWISE
      BEGIN {all non-About items in this menu are desktop items, }
           { for example, DA's, other apps, documents, etc.}
         GetMenuItemText(GetMenuHandle(mApple), menuItem,
                        itemName);
         daRefNum := OpenDeskAcc(itemName);
      END;
    END; {of CASE}
END;
```

When the user chooses an item other than your application's About command from the Apple menu, your application should call the `OpenDeskAcc` function. The `OpenDeskAcc` function prepares to open the desktop object chosen by the user; for example, if the user chose a document created by the TeachText application, the `OpenDeskAcc` function schedules the TeachText application for execution (or prepares to open it if it isn't already open) and returns to your application. On your application's next call to `WaitNextEvent`, your application receives a suspend event, and then the Process Manager makes TeachText the foreground application and instructs TeachText to open the chosen document.

Handling the Help Menu

Both the `MenuSelect` and `MenuKey` functions return the `kHMHelpMenuID` constant (–16490) in the high word when the user chooses an appended item from the Help menu. The item number of the appended menu item is returned in the low word of the function result.

The `DoMenuCommand` procedure shown in Listing 3-24 determines which menu command was chosen by the user. If the user chose a command from the Help menu, the `DoMenuCommand` procedure calls the application-defined procedure `MyHandleHelpCommand`. Listing 3-26 shows the application-defined procedure `MyHandleHelpCommand`. This procedure illustrates how the SurfWriter application responds to the user's choice of an item from the application's Help menu. Note that you should use the `HMGetHelpMenuHandle` function, not the `GetMenuHandle` function, to get a handle to your application's Help menu.

Listing 3-26   Responding to the user's choice of a command from the Help menu

```
PROCEDURE MyHandleHelpCommand (menuItem: Integer);
VAR
   myHelpMenuHdl:              MenuHandle;
   origHelpItems, numItems:    Integer;
   myErr:                      OSErr;
BEGIN
   {get handle to your application's Help menu}
   myErr := HMGetHelpMenuHandle(myHelpMenuHdl);
   IF myErr <> noErr THEN
      EXIT(MyHandleHelpCommand);
   {count the number of items in the Help menu}
   numItems := CountMItems(myHelpMenuHdl);
   origHelpItems := numItems – kNumMyHelpItems;
   IF menuItem > origHelpItems THEN
   BEGIN {user chose an item added by this application}
      {adjust this application's global variables that hold item }
      { numbers of the menu items that this application appended}
      gMyHelpItem1 := origHelpItems +1;
```

```
      gMyHelpItem2 := origHelpItems +2;
      MyHelp(menuItem);
   END;
END;
```

Apple reserves the right to change the number of standard items in the Help menu. To determine the number of items in the Help menu, call the CountMItems function.

Handling a Size Menu

Your application can provide a Size menu to let the user choose various sizes of a font. Your Size menu should also provide the user with a method for specifying a size that isn't currently listed in the menu. For example, you can choose to provide an Other command that displays a dialog box allowing the user to choose a different font size. If the user chooses a font size not already in the menu, add a checkmark to the Other menu command and add the chosen size in parentheses to the text of the Other command.

Your application should outline font sizes to indicate which sizes are directly provided by the current font. For bitmapped fonts, outline only those sizes that actually exist in the Fonts folder. For TrueType fonts, outline all sizes that the TrueType font supports.

Your application should indicate the current font size to the user by placing a checkmark next to the text of the menu item that lists the current font size. If the current selection contains more than one font size, place a dash next to the name of each font size that the selection contains. ("Changing the Mark of Menu Items" on page 3-61 explains how to add marks to and remove marks from menu items.)

Figure 3-35 shows a Size menu as it appears after the user chooses a new font size of 31 by using the Other command. In Figure 3-35 the sizes 9, 10, 12, 18, 24, and 36 are the standard sizes provided by the application. Your application should place a checkmark next to the Other command to indicate that the current font size is a size other than a standard size. If the selection contains only one nonstandard size, include the size of the font in parentheses following the text Other. In Figure 3-35 the current selection contains a nonstandard size of 31, so the application places the checkmark next to the Other command and includes 31 in parentheses following the Other text. If the selection contains multiple nonstandard sizes, include the text Mixed in parentheses following the word Other. If the selection contains one or more standard sizes and only one nonstandard size, place a dash next to each standard size that the selection contains and place a dash next to the Other command with the nonstandard size included in paretheses in the text of the Other command.

Figure 3-35   A Size menu with user-specified size added
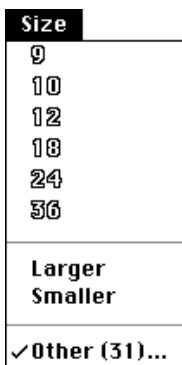
When the user chooses the Other command, you should display the current font size in a dialog box and allow the user to choose a new size. Figure 3-16 on page 3-28 shows a sample dialog box an application might display in response to the user's choice of the Other command.

You should always specify the text of the Other command in the plain font style (as shown in Figure 3-35) and never outlined, regardless of whether the current font is a TrueType font that supports that size or a bitmapped font that exists at that size in the Fonts folder.

Listing 3-27 shows an application-defined procedure that handles the user's choice of an item in the Size menu shown in Figure 3-35.

Listing 3-27   Handling the Size menu

```
PROCEDURE MyHandleSizeCommand (menuItem: Integer);
VAR
   numItems:      Integer;
   addItem:       Boolean;
   itemString:    Str255;
   itemStyle:     Style;
   sizeChosen:    LongInt;
BEGIN
   numItems := CountMItems(GetMenuHandle(mSize));
   IF menuItem = numItems THEN
   BEGIN                              {user chose Other command}
        {display a dialog box to allow the user to choose any }
        { size. If the user-specified size is not in the menu, }
        { add a checkmark to the Other command and add the }
        { new font size to the text of the Other command}
      MyDisplayOtherBox(sizeChosen);
   END
   ELSE
```

```
    BEGIN
        IF (menuItem = (numItems -2)) OR
            (menuItem = (numItems -3)) THEN
            DoMakeLargerOrSmaller(menuItem, sizeChosen)
        ELSE
        BEGIN                {user chose size displayed in the menu}
            {remove checkmark or dashes from menu items showing }
            { previous size}
            MyRemoveMarksFromSizeMenu;
            {add checkmark to menu item of new current size}
            CheckItem(GetMenuHandle(mSize), menuItem, TRUE);
            sizeChosen := MyItemToSize(menuItem);
        END;
    END;
    {update the document's state or the user's selection as needed}
    MyResizeSelection(sizeChosen);
END;
```

If the user chooses an item from the Size menu, the MyHandleSizeCommand procedure first counts the current number of items in the menu. If the user chooses the last item in the menu (the Other command), the procedure displays a dialog box like the one shown in Figure 3-16 on page 3-28 to let the user choose a size other than the ones currently shown in the menu. The application-defined function MyDisplayOtherBox also adds a checkmark to the Other command if the user chose a new size, adds the new size to the text of the Other command, and returns the chosen size in the sizeChosen variable.

If the user chose the Larger or Smaller command from the Size menu, the code calls an application-defined routine, DoMakeLargerOrSmaller, to perform the requested action. The DoMakeLargerOrSmaller procedure also adds a checkmark and adds the new size to the text of the Other command if the new size does not match any size in the menu. The procedure returns the chosen size in the sizeChosen variable.

If the user chose any size currently displayed in the menu, the MyHandleSizeCommand procedure adjusts the marking character of the menu items appropriately. The code removes the checkmark from the previous menu item and adds a checkmark to the menu item representing the new size chosen by the user. The code uses an application-defined function, MyItemToSize, to map the item number of the chosen menu item to a given size and returns this size in the sizeChosen variable.

The code then uses the application-defined procedure MyResizeSelection to update the document's state and resize the user's selection, if any, to the chosen size.

Accessing Menus From a Dialog Box

In System 7, the Menu Manager or your application can allow the user to access selected menus in the menu bar while interacting with an alert box or a modal dialog box. This allows users to make menu selections while your application is displaying an alert box or a modal dialog box. For example, a user might want to turn on Balloon Help for

assistance in figuring out how to respond to an alert box. Similarly, if the modal dialog box contains several editable text fields, the user might find it simpler to copy text from one text field and paste it into another. Figure 3-36 shows a modal dialog box with an editable text field. Note that only the Edit and Help menus are enabled and all other menus are disabled. This gives the user access to editing commands and also to Balloon Help.

Note

In System 6, user access to menus in the menu bar is prohibited from an alert box or a modal dialog box unless your application specifically allows it. For example, in System 6, your application must provide a filter procedure to replace the standard filter procedure if you want to support the keyboard equivalents of the standard Edit menu commands in a modal dialog box. In System 7, you can let the Menu Manager enable these commands for you. ◆

Figure 3-36 Menu access from a modal dialog box

When your application displays a modeless or movable modal dialog box, your application should adjust its menus as appropriate for that dialog box. For example, when a movable modal dialog box is the frontmost window, your application should enable the Apple menu, enable the Edit menu if your dialog box contains an editable text item, enable or disable any other menus as needed, and disable any items it added to the Help menu if the user can't perform those actions while the dialog box is displayed.

When your application displays an alert box, system software automatically disables all of your application's menus except for the Help menu (in which all items are disabled except for the Show Balloons/Hide Balloons command).

When your application displays a modal dialog box, your application should also enable and disable its menus as appropriate. For example, you should enable the Edit menu if your dialog box contains an editable text item and disable any items it added to the Help menu if the user can't perform those actions while the dialog box is displayed. If your application handles access to the menu bar from a modal dialog box, it should disable the Apple menu or the first item in the Apple menu.

If your application does not specifically handle access to the menu bar from an alert box or a modal dialog box, in some cases the Menu Manager automatically disables the appropriate menus for you, as described in the following paragraphs.

When your application displays an alert box or a modal dialog box (that is, a window of type dBoxProc), the Menu Manager (in conjunction with the Dialog Manager) always appropriately adjusts the system-handled menus and performs these actions:

1. Disables all menu items in the Help menu except the Show Balloons (or Hide Balloons) command, which it enables.

2. Disables all menu items in the Application menu.

3. Enables the Keyboard menu if it appears in the menu bar, except for the About Keyboards command, which it disables.

In addition, if your application then calls the ModalDialog procedure, the Menu Manager (in conjunction with the Dialog Manager) performs two other actions:

4. Disables all of your application's menus.

5. Enables commands with the standard keyboard equivalents Command-X, Command-C, and Command-V if the modal dialog box contains a visible and active editable text field. The user can then use either the menu commands or their keyboard equivalents to cut, copy, and paste text. (The menu item having keyboard equivalent Command-X must be one of the first five menu items.)

When the user dismisses the modal dialog box, the Menu Manager restores all menus to the state they were in prior to the appearance of the modal dialog box.

In some cases actions 4 and 5 do not occur when you call ModalDialog. The enabling and disabling described in steps 4 and 5 do not occur if any of these conditions is true:

■ Your application does not have an Apple menu.

■ Your application has an Apple menu, but the menu is disabled when the modal dialog box is displayed.

■ Your application has an Apple menu, but the first item in that menu is disabled when the dialog box is displayed.

Note
If your application already handles access to the menu bar from a modal dialog box and you do not want the automatic menu enabling and disabling provided by System 7 to occur, you should ensure that one or more of those conditions is true when you display a modal dialog box. ◆

When your application displays alert boxes or modal dialog boxes with no editable text items, your application can allow system software to handle menu bar access. In all other cases, your application should handle its own menu bar access.

System software always leaves the Help, Keyboard, and Application menus and their commands available when you display movable modal dialog boxes and modeless dialog boxes. For these types of dialog boxes, you must disable menus as appropriate and handle menu bar access as appropriate given their contents.

When your application displays a movable modal dialog box (a window of type `movableDBoxProc`), your application does not need to adjust the system-handled menus but should disable all its other menus except the Apple menu and—if your movable modal dialog box contains editable text items—the Edit menu. Leave the Apple menu enabled so that the user can use it to open other applications, and leave the Edit menu enabled so that the user can use the Cut, Copy, and Paste commands within the editable text item. (You can also leave your Undo and Clear commands enabled; otherwise, disable all other commands in the Edit menu.)

When your application removes a movable modal dialog box, modeless dialog box, or modal dialog box with editable text items, your application must restore to their previous states any menus that it disabled prior to displaying the dialog box. See the chapter "Dialog Manager" in this book for additional information on dialog boxes.

Writing Your Own Menu Definition Procedure

The Menu Manager uses the menu definition procedure and menu bar definition function to display and perform basic operations on menus and the menu bar. The menu definition procedure performs all the drawing of menu items within a menu and performs all the actions that might differ between one type of menu and another. The menu bar definition function draws the menu bar and performs most of the drawing activities related to the display of menus when the user moves the cursor between menus.

Apple provides a standard menu bar definition function, stored as a resource in the System file. The standard menu bar definition procedure is the `'MBDF'` resource with resource ID 0. When you create your menus and menu bar, by default the Menu Manager uses the standard menu bar definition function to manage them. Although the Menu Manager lets you provide your own menu bar definition function, Apple recommends that you always use the standard menu bar definition function.

The Menu Manager uses the standard menu bar definition function to

■ draw the menu bar

■ clear the menu bar

■ determine if the cursor is in the menu bar or any currently displayed menus

■ calculate the left edges of menu titles

■ highlight a menu title

■ invert the entire menu bar

■ erase the background color of a menu and draw the menu's structure (shadow)

■ save or restore the bits behind a menu

Apple provides a standard menu definition procedure, stored as a resource in the System file. The standard menu definition procedure is the `'MDEF'` resource with resource ID 0. The standard menu definition procedure handles three types of menus: pull-down, pop-up, and hierarchical; it also implements scrolling in menus. When you define your menus, you specify the menu definition procedure that the Menu Manager should use

when managing them. You'll usually want to use the standard definition procedure for your application. However, if you need a feature not provided by the standard menu definition procedure (for example, if you want to include more graphics in your menus), you can write your own menu definition procedure.

The Menu Manager uses the standard menu definition procedure to

■ calculate a menu's dimensions

■ draw the menu items in a menu

■ highlight and unhighlight menu items as the user moves the cursor between them

■ determine which item the user chose from a menu

If you provide your own menu definition procedure, it should also perform these tasks. Your menu definition procedure should also support scrolling in menus and color in menus and provide support for Balloon Help.

If you provide your own menu definition procedure, store it in a resource of type `'MDEF'` and include its resource ID in the description of each menu that uses your own menu definition procedure. If you create a menu using `GetMenu` (or `GetNewMBar`), the Menu Manager reads the menu definition procedure into memory and stores a handle to it in the `menuProc` field of the menu's menu record.

When your application uses `GetMenu` (or `GetNewMBar`) to create a new menu that uses your menu definition procedure, the Menu Manager creates a menu record for the menu and fills in the `menuID`, `menuProc`, `enableFlags`, and `menuData` fields according to the menu's resource description. The Menu Manager also reads in the data for each menu item and stores it as variable data at the end of the menu record. The menu definition procedure is responsible for interpreting the contents of the data. For example, the standard menu definition procedure interprets this data as described in "The Menu Resource" beginning on page 3-151. After reading in a resource description of a menu, the Menu Manager requests the menu definition procedure to calculate the size of the menu and to store these values in the `menuWidth` and `menuHeight` fields of the menu's menu record.

Note that when drawing a menu, the Menu Manager first requests your menu definition procedure to calculate the dimensions (the menu rectangle) of the menu. Next the Menu Manager requests the menu bar definition function to draw the structure (shadow) of the menu and erase the contents of the menu to its background color. Then the Menu Manager requests your menu definition procedure to draw the items in the menu. As the user moves the cursor into and out of menu items, the Menu Manager requests your menu definition procedure to highlight and unhighlight items appropriately. Your menu definition procedure should also determine when to add scrolling indicators to a menu and scroll the menu appropriately when the cursor is in a scrolling item. Your menu definition is responsible for showing and removing any help balloons associated with a menu item.

When the Menu Manager requests your menu definition procedure to perform an action on a menu, it provides your procedure with a handle to its menu record. This allows your procedure to access the data in the menu record and to use any data in the variable data portion of the menu record to appropriately handle the menu items. However, your

menu definition procedure should not assume that the A5 register is properly set up, so your procedure can't refer to any of the QuickDraw global variables.

The Menu Manager passes a value to your menu definition procedure in the message parameter that indicates the action your menu definition procedure should perform. The Menu Manager always passes a handle to the menu record of the menu that the operation should affect in the parameter theMenu. Depending on the requested action, the Menu Manager passes additional information in other parameters.

Listing 3-28 shows how you might declare a menu definition procedure.

Listing 3-28   A sample menu definition procedure

```
PROCEDURE MyMDEF (message: Integer; theMenu: MenuHandle;
                  VAR menuRect: Rect; hitPt: Point;
                  VAR whichItem: Integer);

{any support routines used by the main program of your MDEF }
{ go here}

BEGIN
   CASE message OF
      mDrawMsg:
         MyDrawMenu(theMenu, menuRect);
      mChooseMsg:
         MyChooseItem(theMenu, menuRect, hitPt, whichItem);
      mSizeMsg:
         MySizeTheMenu(theMenu);
      mPopUpMsg:
         MyCalcMenuRectForOpenPopUpBox(theMenu, hitPt, menuRect);
   END;
END;
```

The next sections describe in more detail how your menu definition procedure should respond when it receives the mDrawMsg, mChooseMsg, or mSizeMsg constant in the message parameter. For a complete description of the menu definition procedure and the parameters passed to your procedure by the Menu Manager, see "The Menu Definition Procedure" beginning on page 3-148.

Calculating the Dimensions of a Menu

Whenever the Menu Manager creates a menu or needs to calculate the size of a menu that is managed by your menu definition procedure, the Menu Manager calls your procedure and specifies the mSizeMsg constant in the message parameter, requesting that your procedure calculate the size of the menu.

Listing 3-29 on page 3-90 shows an application-defined support routine,
`MySizeTheMenu`, used by the application's menu definition procedure. After
calculating the height and width of the menu's rectangle, the menu definition
procedure stores the values in the `menuWidth` and `menuHeight` fields of the
menu's menu record.

Listing 3-29 Calculating the size of a menu

```
PROCEDURE MySizeTheMenu(theMenu: MenuHandle);
VAR
   itemDataPtr:    Ptr;
   numItems:       Integer;
BEGIN
   HLock(Handle(theMenu));
   WITH theMenu^^ DO
   BEGIN    {menuData points to title of menu and additional item data}
      itemDataPtr := @menuData;
      {skip past the menu title}
      itemDataPtr := POINTER(ORD4(itemDataPtr)+ itemDataPtr^ +1);
   END;
   numItems := CountMItems(theMenu);
   {calculate the height of the menu--each item's height can vary }
   { according to whether the item has an icon or a script code defined. }
   { The height of the menu should not exceed the height of the }
   { screen minus the menu bar height. }
   { Store the height in the menu's menu record}
   theMenu^^.menuHeight := MyCalcMenuHeight(itemDataPtr, numItems);

   {calculate the width of the menu (the width of the longest item): }
   { for each item calculate the width as }
   { width = iconWidth + markWidth + textWidth + subMenuWidth }
   {         + cmdKeyComboWidth }
   { If an item doesn't have a characteristic, use 0 as the width of }
   { that characteristic. }
   { To calculate the width of item's text, must consider script code and }
   { width of the font. }
   { The width of the menu should not exceed the right or left }
   { boundaries of the screen. }
   { Store the width in the menu's menu record}
   theMenu^^.menuWidth := MyCalcMenuWidth(itemDataPtr, numItems);
   HUnLock(Handle(theMenu));
END;
```

Drawing Menu Items in a Menu

Whenever the user presses the mouse button while the cursor is in the menu title of a menu managed by your menu definition procedure, the Menu Manager calls the menu bar definition function to highlight the menu title, draw the structure of the menu, and erase the contents of the menu to its background color. The Menu Manager then calls your menu definition procedure and specifies the mDrawMsg constant in the message parameter, requesting that your procedure draw the menu items. When your menu definition procedure receives this constant, it should draw the menu items of the menu specified by the parameter theMenu inside the rectangle specified by the menuRect parameter. The Menu Manager sets the current graphics port to the Window Manager port before calling your menu definition procedure. Your menu definition procedure can determine how to draw the menu items by examining the data in the menu record.

If your menu definition procedure supports color menus, your procedure should check the application's menu color information table for the colors to use to draw each item. If the application's menu color information table contains a color entry for an item, draw the item using that color. If the table does not contain an item entry for a particular item, use the default item color defined in the menu title entry. If a menu title entry doesn't exist, use the default item color defined in the menu bar entry. If the menu bar entry doesn't exist, draw the item using black on white.

If your menu definition procedure supports scrolling menus, it should insert scrolling indicators if necessary when drawing the menu items.

Listing 3-30 shows an application-defined support routine, MyDrawMenu, used by the application's menu definition procedure. The MyDrawMenu procedure draws each item in the menu, according to the item's defined characteristics. Disabled items should be drawn using the colors returned by the GetGray function. Pass the RGB color of the item's background in the bkgnd parameter to the GetGray function; pass the RGB color of the item's enabled text in the fgnd parameter. The GetGray function returns TRUE if there's an available color between the two specified colors and returns in the fgnd parameter the color in which you should draw the item.

Listing 3-30   Drawing menu items

```
PROCEDURE MyDrawMenu(theMenu: MenuHandle; menuRect: Rect);
VAR
    numItems:       Integer;
    itemRect:       Rect;
    item:           Integer;
    currentOffset:  LongInt;
    nextOffset:     LongInt;
BEGIN
    numItems := CountMItems(theMenu);
    currentOffset := 0;
    nextOffset := 0;
```

```
    FOR item := 1 TO numItems DO
    BEGIN
    {calculate the enclosing rectangle for this item}
        itemRect := MyCalcItemRect(item, menuRect, currentOffset, nextOffset);
    {draw the item--index into the item-specific data from the menu record }
    { to get the characteristics of this menu item and draw the item }
    { according to its defined characteristics. For example, draw the item's }
    { text in its defined style & font of its defined script, draw any icon, }
    { mark, submenu indication, or keyboard equivalent, and draw each }
    { characteristic of the item according to its color entry in the menu's }
    { menu color information table. }
    { Draw disabled items in gray--use the GetGray function to return the }
    { appropriate color. Also draw dividers using the gray color }
    { returned by GetGray}
        MyDrawTheItem(item, itemRect, menuRect, currentOffset);
    END;
    {if your menu supports scrolling, insert scrolling indicators if needed}
    MyInsertScrollingArrows(menuRect);
END;
```

Determining Whether the Cursor Is in an Enabled Menu Item

Whenever the user drags the cursor into or out of a menu item of a displayed menu managed by your menu definition procedure, the Menu Manager calls your procedure and specifies the mChooseMsg constant in the message parameter, requesting that your procedure determine whether the cursor is in a menu item and that your procedure highlight or unhighlight the menu item as appropriate. When your menu definition procedure receives this constant, it should use the menu rectangle specified in the menuRect parameter, the mouse location specified in the hitPt parameter, and the item number specified in the whichItem parameter to determine the proper action to take.

To see whether the user chose an enabled item, your menu definition procedure should determine whether the specified mouse location is inside the rectangle specified by the menuRect parameter, and, if so, it should check whether the menu is enabled. If the menu is enabled, your menu definition procedure should determine whether the mouse location specified in the hitPt parameter is in an enabled menu item.

If the mouse location is in an enabled menu item, your menu definition procedure should unhighlight the item specified by the whichItem parameter, highlight the new item, and return the new item number in whichItem.

If the mouse location isn't in an enabled menu item, your menu definition procedure should unhighlight the item specified by the whichItem parameter and return 0 in the whichItem parameter.

When your menu definition procedure draws a menu item in its highlighted state in a color menu, it should reverse the background color and the item color and then draw the menu item. When your menu definition procedure needs to return a menu item to its normal (unhighlighted) state, it should reset the background color and item color of that menu item and draw the menu item.

If your menu definition procedure supports scrolling menus, it should scroll the menu when the user moves the cursor into the area of the indicator, or when the cursor is directly above or below the menu. If the user can scroll the menu up (by dragging the cursor past the last item to view more items), place a downward-pointing triangular indicator in place of the last item in the menu. If the user can scroll the menu down (by dragging the cursor past the first item to view the items originally at the top of the menu), place an upward-pointing triangular indicator in place of the first item in the menu.

For all menus, your menu definition procedure should set the global variable MenuDisable appropriately each time a new item is highlighted. Set MenuDisable to the menu ID and item number of the last menu item chosen, whether or not it's disabled. The MenuChoice function uses the value in MenuDisable to determine if a chosen menu item is disabled.

Listing 3-31 shows an application-defined support routine, MyChooseItem, used by the application's menu definition procedure. This routine determines which item, if any, the point specified by the hitPt parameter is in. If the item is in an enabled menu item that is different from the previous item, the MyChooseItem procedure unhighlights the old item and highlights the new item. However, the MyChooseItem procedure does not highlight the new item if the item is in a divider or disabled item.

The procedure also removes any help balloons as appropriate and, if Balloon Help is turned on, displays any help balloon of the new item (for any item other than a divider or scrolling indicator). The MyChooseItem procedure returns the item number of the new item in the whichItem parameter or returns 0 if no item is chosen. Although not shown in the listing, if the item is a disabled item, the procedure returns 0 in the whichItem parameter and sets the MenuDisable global variable to the menu ID and item number of the disabled item.

Listing 3-31   Choosing menu items

```
PROCEDURE MyChooseItem (theMenu: MenuHandle; menuRect: Rect; hitPt: Point;
                        VAR whichItem: Integer);
VAR
   oldWhichItem:          Integer;
   MenuChoicePtr:         ^LongInt;
   numItems, item, max:   Integer;
   itemChosen:            Integer;
   inScroll:              Integer;
   currentOffset:         LongInt;
   nextOffset:            LongInt;
```

```
BEGIN
   oldWhichItem := whichItem;
   whichItem := 0;
   itemChosen := 0;
   MenuChoicePtr := POINTER(kLowMemMenuDisable);
   numItems := CountMItems(theMenu);
   {find out whether the hitPt is in an item's rectangle, and if so, }
   { determine which item}
   item := 1;
   max := numItems + 1;
   currentOffset := 0;
   nextOffset := 0;
   REPEAT
      itemRect := MyCalcItemRect(item, menuRect, currentOffset, nextOffset);
      IF PtInRect(hitPt, itemRect) THEN {hitPt is in this item}
         itemChosen := item;
      item := item + 1;
   UNTIL (item = MAX) OR (itemChosen <> 0);
   IF itemChosen = 0 THEN
   BEGIN {the mouse isn't in any item of this menu;unhighlight previous item}
      MyNotInMenu(menuRect, oldWhichItem);
   END
   ELSE
   BEGIN {the mouse is in this menu item. }
         { First see if a previous item was highlighted}
      IF ((oldWhichItem <> 0) AND (oldWhichItem <> itemChosen)) THEN
      BEGIN
         {a previous item was highlighted--unhighlight it}
         itemRect := MyCalcOldItemRect(oldWhichItem, menuRect);
         IF HMGetBalloons THEN {if Balloon Help is on then }
            HMRemoveBalloon;{ remove any balloon that might be showing}
         MyHighlightItem(itemRect, oldWhichItem, FALSE);
      END;
      IF HMGetBalloons and MyIsItemDivider(itemChosen) THEN
         {Balloon Help is on and item is divider}
         HMRemoveBalloon;{remove any balloon that might be showing}
      IF MyIsItemEnabled(itemChosen) THEN
      BEGIN
         {the item is enabled, so highlight the item the cursor is in}
         itemRect := MyCalcNewItemRect(itemChosen, menuRect, currentOffset);
         {the highlighting routine must also support scrolling correctly }
         { (if the cursor is in a scrolling item, don't highlight the item)}
         inScroll := MyIsScrollItem(itemChosen);
```

```
    MyHighlightItem(itemRect, itemChosen, inScroll);
    IF HMGetBalloons AND inScroll THEN
       HMRemoveBalloon    {remove any balloon that might be showing}
    ELSE
    BEGIN {display help balloon for this item, if any}
       IF HMGetBalloons THEN


       BEGIN
          IF StillDown THEN {mouse button is still down in this item}
             {this routine sets up the needed parameters and then }
             { calls HMShowMenuBalloon}
             MyShowMenuBalloon(itemChosen, itemRect);
       END;
    END;
  END;
END;
```

Menu Manager Reference

This section describes the data structures and routines of the Menu Manager. It also describes various resources, including the resources you can use to create your menus and menu bar, the 'MBAR' and 'MENU' resources.

Data Structures

This section describes the menu record, menu list, and menu color information table. The Menu Manager maintains information about the menus in your application in menu records. The Menu Manager maintains information about all the menus in a menu bar in a data structure called the *menu list*.

The Menu Manager stores color information about your application's menus in a menu color information table. You can add entries to your application's menu color information table if you want to use colors other than the default colors for your menu bar or menus. You can add entries to this table by using the SetMCEntries procedure or by providing 'mctb' resources.

The Menu Record

A menu record contains information about a single menu. Your application should never manipulate or access the fields of a menu record; instead your application should use

Menu Manager routines to create and manage the menus in your application. To refer to a menu, use a handle to the menu's menu record.

The `MenuInfo` data type defines the menu record. The `MenuHandle` data type is a handle to a menu record.

```
TYPE  MenuPtr    = ^MenuInfo;   {pointer to a menu record}
      MenuHandle = ^MenuPtr;    {handle to a menu record}
```

Here is the structure of a menu record:

```
TYPE   MenuInfo =                      {menu record}
       RECORD
           menuID:        Integer;     {number that identifies the menu}
           menuWidth:     Integer;     {width (in pixels) of the menu}
           menuHeight:    Integer;     {height (in pixels) of the menu}
           menuProc:      Handle;      {menu definition procedure}
           enableFlags:   LongInt;     {indicates whether menu and }
                                       { menu items are enabled}
           menuData:      Str255;      {title of menu}
           {itemDefinitions}           {variable-length data that }
                                       { defines the menu items}
       END;
```

Field descriptions

menuID   A number that identifies the menu. Each menu in your application
                       must have a unique menu ID. Your application specifies the menu
                       ID when you create the menu. Thereafter you can use the menu ID
                       and the GetMenuHandle function to get a handle to the menu's
                       menu record.

                       When you define hierarchical menus, you must use a number from
                       1 through 235 for the menu ID of a submenu of an application; use a
                       number from 236 through 255 for the submenu of a desk accessory.

menuWidth   The horizontal dimensions of the menu, in pixels.

menuHeight   The vertical dimensions of the menu, in pixels.

menuProc   A handle to the menu definition procedure of the menu. The Menu
                       Manager uses this menu definition procedure to draw the menu.

enableFlags   A value that represents the enabled state of the menu title and
                       the first 31 items in the menu. All menu items greater than 31
                       are enabled by default and can be disabled only by disabling the
                       entire menu.

menuData   A string that defines the title of the menu. Although the menuData
                       field is defined by the data type Str255 in the MenuInfo data
                       structure, the Menu Manager allocates only the storage necessary
                       for the title: the number of characters in the title of the string plus 1.

itemDefinitions
                       Variable-length data that defines the characteristics of each menu
                       item in the menu. If the menu uses the standard menu definition
                       procedure, this data can be conceptually defined in this manner:

```
itemData: ARRAY[1..X] OF
    itemString:  String; {text of menu item}
    itemIcon:    Byte;   {icon number minus 256}
```

```
            itemCmd:     Char;    {keyboard equivalent or }
                                  { value ($1B) indicating }
                                  { item has a submenu, or }
                                  { ($1C) if item has }
                                  { a script code, or }
                                  { ($1D) if item's 'ICON' }
                                  { should be reduced, or }
                                  { ($1E) if item has an }
                                  { 'SICN' icon}
            itemMark:    Char;    {marking character or }
                                  { menu ID of submenu}
            itemStyle:   Style;   {style of menu text}
         endMarker:      Byte;    {contains 0 if no }
                                  { more menu items}
```

The menu definition procedure maintains the information about the menu items. You typically define your menu items in 'MENU' resources, and the Menu Manager stores information describing your items in the menu's menu record.

Your application should not directly change the values of any fields in a menu record. Use Menu Manager routines to change the characteristics of menu items or to make other changes to a menu.

The Menu List

The menu list contains information about the menus in a menu bar, about submenus, and about pop-up menus. A menu list contains handles to the menu records of zero, one, or more menus and contains other information that the Menu Manager uses to manage menus.

The InitMenus procedure creates the current menu list of an application. The current menu list contains handles to the menu records of all menus currently in the menu bar and handles to the menu records of any submenus or pop-up menus inserted into the menu list by your application. The menu bar shows the titles, in order, of all menus (other than submenus or pop-up menus) in the menu list.

The initial menu list created by InitMenus does not contain handles to any menus. The Menu Manager dynamically allocates storage in a menu list as menus are added to and deleted from the menu list.

Your application should not directly change or access the information in a menu list. You should use Menu Manager routines to create a menu list and to add menus to or remove menus from the current menu list.

You typically define your application's menu bar in an 'MBAR' resource and create a menu list using the GetNewMBar function. The GetNewMBar function returns a handle to a menu list. You can set the current menu list to the menu list returned by GetNewMBar using the SetMenuBar procedure.

The structure of the menu list is private to the Menu Manager. For conceptual purposes, however, its general structure is defined here.

```
TYPE  DynamicMenuList =
      RECORD
          lastMenu:   Integer;     {offset to last pull-down menu}
          lastRight:  Integer;     {pixel location of right edge }
                                   { of rightmost menu in menu bar}
          mbResID:    Integer;     {upper 13 bits are the resource ID of menu }
                                   { bar defn function, low 3 bits the variant}
          menu:       ARRAY[1..X] {variable array with one record for }
                      OF MenuRec; { each menu}
          lastHMenu:  Integer;     {offset to last submenu or pop-up menu}
          menuTitleSave:           {handle to bits behind inverted menu title}
                      pixMapHandle;
          hMenu:      ARRAY[1..Y] {variable array with one record for }
                      OF HMenuRec;{ each submenu or pop-up menu}
      END;
```

The Menu Manager dynamically allocates the records that contain handles to the menu records of menus in the menu bar, submenus, and pop-up menus. These records can be defined conceptually as the MenuRec and HMenuRec data types. The Menu Manager uses a data structure similar to that of the MenuRec data type to store information about pull-down menus in the menu list.

```
    TYPE  MenuRec =
          RECORD
              menuOH:    MenuHandle; {handle to menu's menu record}
              menuLeft:  Integer;    {pixel location of left edge }
                                     { of this menu}
          END;
```

The Menu Manager stores information about submenus and pop-up menus at the end of a menu list in a data structure similar to that of the HMenuRec data type.

```
    TYPE  HMenuRec =
          RECORD
              menuHOH:   MenuHandle; {handle to menu's menu record}
              reserved:  Integer;    {reserved}
          END;
```

The Menu Color Information Table Record

Your application's **menu color information table** defines the standard color for the menu bar, titles of menus, text and characteristics of menu items, and background color of a displayed menu. If you do not add any entries to this table, the Menu Manager draws your menus using the default colors, black on white. You can add colors to your

menus by adding entries to your application's menu color information table by using Menu Manager routines or by defining these entries in an `'mctb'` resource. Note that the menu color information table uses a format that is different from the standard color table format.

The Menu Manager maintains information about an application's menu color information table as an array of menu color entry records.

```
TYPE  MCTable = ARRAY[0..0] OF MCEntry;         {menu color table}
      MCTablePtr = ^MCTable;      {pointer to a menu color table}
      MCTableHandle = ^MCTablePtr;{handle to a menu color table}
```

A menu color entry is defined by the `MCEntry` data type.

```
TYPE  MCEntry =                     {menu color entry}
      RECORD
          mctID:      Integer;    {menu ID or 0 for menu bar}
          mctItem:    Integer;    {menu item number or 0 for }
                                  { menu title}
          mctRGB1:    RGBColor;   {usage depends on mctID and }
                                  { mctItem}
          mctRGB2:    RGBColor;   {usage depends on mctID and }
                                  { mctItem}
          mctRGB3:    RGBColor;   {usage depends on mctID and }
                                  { mctItem}
          mctRGB4:    RGBColor;   {usage depends on mctID and }
                                  { mctItem}
          mctreserved:Integer;    {reserved}
      END;


      MCEntryPtr = ^MCEntry;      {pointer to a menu color entry}
```

The first two fields of a menu color entry record, `mctID` and `mctItem`, define whether the entry is a menu bar entry, a menu title entry, or a menu item entry. The following four fields specify color information for whatever type of entry the `mctID` and `mctItem` fields describe. The value of the `mctID` field in the last entry in a menu color information table is –99, and the rest of the fields of the last entry are reserved. The Menu Manager automatically creates the last entry in a menu color information table; your application should not use the value –99 as the menu ID of a menu if you wish to add a menu color entry for it.

The Menu Manager creates your application's menu color information table the first time your application calls `InitMenus` or `InitProcMenu`. It creates the menu color information table as initially empty except for the last entry, which indicates the end of the table.

Table 3-7 shows how the Menu Manager interprets the `mctID` and `mctItem` fields for each type of menu color entry in a menu color information table.

Table 3-7   Color information for menu entries

	ID	Item	RGB1	RGB2	RGB3	RGB4
Menu bar	0	0	Default menu title color	Default background color of menus	Default item color	Default bar color
Menu title	$N<>0$	0	Menu title color	Bar color	Default item color	Background color of menu
Menu item	$N<>0$	$M<>0$	Mark color	Item text color	Keyboard equivalent color	Background color of menu
Last entry	−99	Reserved	Reserved	Reserved	Reserved	Reserved

A **menu bar entry** is defined by a menu color entry record that contains 0 in both the `mctID` and `mctItem` fields. You can define only one menu bar entry in a menu color information table. If you don't provide a menu bar entry for your application's menu color information table, the Menu Manager uses the standard menu bar colors (black text on a white background), and it uses the standard colors for the other menu elements. You can provide a menu bar entry to specify default colors for the menu title, the background of a displayed menu, the items in a menu, and the menu bar. The color information fields for a menu bar entry are interpreted as follows:

■ `mctRGB1` specifies the default color for menu titles. If a menu doesn't have a menu title entry, the Menu Manager uses the value in this field as the color of the menu title.

■ `mctRGB2` specifies the default color for the background of a displayed menu. If a menu doesn't have a menu title entry, the Menu Manager uses the value in this field as the color of the menu's background when it is displayed.

■ `mctRGB3` specifies the default color for the items in a displayed menu. If a menu item doesn't have a menu item entry or a default color defined in a menu title entry, the Menu Manager uses the value in this field as the color of the menu item.

■ `mctRGB4` specifies the default color for the menu bar. If a menu doesn't have a menu bar entry (and doesn't have any menu title entries), the Menu Manager uses the standard colors for the menu bar.

A **menu title entry** is defined by a menu color entry record that contains a menu ID in the `mctID` field and 0 in the `mctItem` field. You can define only one menu title entry for each menu. If you don't provide a menu title entry for a menu in your application's menu color information table, the Menu Manager uses the colors defined by the menu bar entry. If a menu bar entry doesn't exist, the Menu Manager uses the standard colors

(black on white). You can provide a menu title entry to specify a color for the title and background of a specific menu and a default color for its items. The color information fields for a menu title entry are interpreted as follows:

■ `mctRGB1` specifies the color for the menu title of the specified menu. If a menu doesn't have a menu title entry, the Menu Manager uses the default value defined in the menu bar entry.

■ `mctRGB2` specifies the default color for the menu bar. If a menu color information table doesn't have a menu bar entry, the Menu Manager uses the value in this field as the color of the menu bar. If a menu bar entry already exists, the Menu Manager replaces the value in the `mctRGB2` field of the menu title entry with the value defined in the `mctRGB4` field of the menu bar entry.

■ `mctRGB3` specifies the default color for the items in the menu. If a menu item doesn't have a menu item entry or a default color defined in a menu bar entry, the Menu Manager uses the value in this field as the color of the menu item.

■ `mctRGB4` specifies the color for the background of the menu.

A **menu item entry** is defined by a menu color entry record that contains a menu ID in the `mctID` field and an item number in the `mctItem` field. You can define only one menu item entry for each menu item. If you don't provide a menu item entry for an item in your application's menu color information table, the Menu Manager uses the colors defined by the menu title entry (or by the menu bar entry if the menu containing the item doesn't have a menu title entry). If neither a menu title entry nor a menu bar entry exists, the Menu Manager draws the mark, text, and keyboard equivalent in black. You can provide a menu item entry to specify a color for the mark, text, and keyboard equivalent of a specific menu item. The color information fields for a menu item entry are interpreted as follows:

■ `mctRGB1` specifies the color for the mark of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry.

■ `mctRGB2` specifies the color for the text of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry. The Menu Manager also draws a black-and-white icon of a menu item using the same color as defined by the `mctRGB2` field. (Use a `'cicn'` resource to provide a menu item with a color icon.)

■ `mctRGB3` specifies the color for the keyboard equivalent of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry.

■ `mctRGB4` specifies the color for the background of the menu. If the menu color information table doesn't have a menu title entry for the menu this item is in, or doesn't have a menu bar entry, the Menu Manager uses the value in this field as the background color of the menu. If a menu title entry already exists, the Menu Manager replaces the value in the `mctRGB4` field of the menu item entry with the value defined in the `mctRGB4` field of the menu title entry (or with the `mctRGB2` field of the menu bar entry).

You can use the GetMCInfo function to get a copy of your application's menu color information table and the SetMCEntries procedure to set entries of your application's menu color information table, or you can provide 'mctb' resources that define the color entries for your menus.

The GetMenu, GetNewMBar, and ClearMenuBar routines can also modify the entries in the menu color information table. The GetMenu function looks for an 'mctb' resource with a resource ID equal to the value in the menuID parameter. If it finds one, it adds the entries to the application's menu color information table.

The GetNewMBar function builds a new menu color information table when it creates the new menu list. If you want to save the current menu color information table, call GetMCInfo before calling GetNewMBar.

The ClearMenuBar procedure reinitializes both the current menu list and the menu color information table.

Menu Manager Routines

The Menu Manager includes routines for creating menus, changing the characteristics of menu items, and handling user choice of menu commands. The Menu Manager also provides routines for adding items to and deleting items from menus, counting the number of items in a menu, getting a handle to a menu's menu record, disposing of menus, calculating the dimensions of a menu, highlighting the menu bar, and managing entries in your application's menu color information table.

Some Menu Manager routines can be accessed using more than one spelling of the routine's name, depending on the interface files supported by your development environment. For example, GetMenuHandle is also available as GetMHandle. Table 3-8 provides a mapping between the previous name of a routine and its new equivalent name.

Table 3-8   Mapping between new and previous names of Menu Manager routines

New name	Previous name
AppendResMenu	AddResMenu
DeleteMCEntries	DelMCEntries
DeleteMenuItem	DelMenuItem
DisposeMCInfo	DispMCInfo
GetMenuHandle	GetMHandle
GetMenuItemText	GetItem
InsertMenuItem	InsMenuItem
SetMenuItemText	SetItem

Initializing the Menu Manager

You can use the `InitMenus` procedure to initialize the Menu Manager.

You can use the `InitProcMenu` procedure to set the current menu list so that it uses a custom menu bar definition function if necessary.

InitMenus

The `InitMenus` procedure allocates space for your application's current menu list in your application's heap. Your application needs to call `InitMenus` only once to initialize the Menu Manager and the current menu list for your application.

```
PROCEDURE InitMenus;
```

DESCRIPTION

The `InitMenus` procedure creates the current menu list with no menus, submenus, or pop-up menus. `InitMenus` also creates your application's menu color information table. After allocating the menu color information table, `InitMenus` looks for an `'mctb'` resource with resource ID 0. You can provide an `'mctb'` resource with a resource ID of 0 as one of your application's resources if you want to use colors other than the default colors for your application's menu bar and menus. If `InitMenus` finds and successfully loads an `'mctb'` resource, it adds the information contained in that resource to the menu color information table (using `SetMCEntries`).

The `InitMenus` procedure also draws an empty menu bar.

SPECIAL CONSIDERATIONS

Your application must initalize QuickDraw, the Font Manager, and the Window Manager (using the `InitGraf`, `InitFonts`, and `InitWindows` procedures) before initializing the Menu Manager.

SEE ALSO

To set up the menus for your application's menu bar, use `GetNewMBar` and `SetMenuBar`, described on page 3-111 and page 3-112, respectively. You can also add menus to the current menu list using the `InsertMenu` procedure, described on page 3-108.

To remove all menus from the current menu list, use the `ClearMenuBar` procedure, described on page 3-110.

If your application uses its own menu bar definition function, use the `InitProcMenu` procedure to set the `mbResID` field of the current menu list to the resource ID of your custom `'MBDF'` resource.

See "The Menu Color Information Table Resource" on page 3-155 for a description of the `'mctb'` resource.

See the chapter "Window Manager" in this book for a description of the `InitWindows` procedure. See *Inside Macintosh: Imaging* and *Inside Macintosh: Text* for descriptions of the `InitGraf` and `InitFonts` procedures.

InitProcMenu

Apple recommends that you use the standard menu bar definition function. However, if your application provides its own menu bar definition function, use the `InitProcMenu` procedure to set the `mbResID` field of the current menu list to the resource ID of your custom `'MBDF'` resource.

```
PROCEDURE InitProcMenu (resID: Integer);
```

resID   The resource ID of your application's menu bar definition function in the upper 13 bits of this parameter; the variant in the lower 3 bits. You must use a resource ID greater than $100.

For resources of type `'MBDF'`, Apple reserves resource IDs $000 through $100 for its own use.

DESCRIPTION

The `InitProcMenu` procedure creates the current menu list if it hasn't already been created by a previous call to `InitMenus`. The `InitProcMenu` procedure stores the resource ID that you specify in the `mbResID` field of the current menu list. The Menu Manager uses the menu bar definition function referred to in this field to draw the menu bar and to perform basic operations on menus.

SPECIAL CONSIDERATIONS

The resource ID of your application's menu bar definition function is maintained in the current menu list until your application next calls `InitMenus`; `InitMenus` initializes the `mbResID` field with the resource ID of the standard menu bar definition function. This can affect applications such as development environments that control other applications that may call `InitMenus`.

SEE ALSO

See the description of the `InitMenus` procedure on page 3-103; you should use `InitMenus` if your application uses the standard menu bar definition function.

Creating Menus

You can use the `NewMenu` or `GetMenu` function to create a pull-down menu, although
you usually create all the menus in your menu bar at once by providing an `'MBAR'`
resource and using the `GetNewMBar` function. See "Getting and Setting the Menu Bar"
on page 3-112 for information on creating a menu bar. You typically use the `NewMenu` or
`GetMenu` function to create submenus or pop-up menus.

The `NewMenu` function creates a menu with the specified title, assigns it the specified
menu ID, and creates a menu record for the menu. Use `AppendMenu`,
`InsertMenuItem`, `AppendResMenu`, or `InsertResMenu` to add items to menus you
create with `NewMenu`.

The `GetMenu` function creates a menu with the title, items, and characteristics defined in
a specified `'MENU'` resource.

Both `NewMenu` and `GetMenu` allocate space in your application's heap for the menu
record and return a handle to the menu's newly created menu record.

To add menus created by `NewMenu` or `GetMenu` to the current menu list, use the
`InsertMenu` procedure. To update the menu bar with any new menu titles, use
`DrawMenuBar`.

NewMenu

You can use the `NewMenu` function to create an empty menu with a specified title and
menu ID. In most cases you should store information about your menus (such as their
titles, items, and characteristics) in resources; use the `GetMenu` or `GetNewMBar` function
to create menus from resource definitions.

```
FUNCTION NewMenu (menuID: Integer; menuTitle: Str255): MenuHandle;
```

menuID   The menu ID of the menu. (Note that this is not the resource ID of a
              `'MENU'` resource.) The menu ID is a number that identifies the menu. Use
              positive menu IDs for menus belonging to your application. Use negative
              menu IDs for desk accessories (except for submenus of a desk accessory).
              Submenus must have menu IDs from 1 through 255. For submenus of an
              application, use menu IDs from 1 through 235; for submenus of a desk
              accessory, use menu IDs from 236 through 255. Apple reserves the menu
              ID of 0.

menuTitle   The title of the new menu. Note that in most cases you should store
              the titles of menus in resources, so that your menu titles can be more
              easily localized.

DESCRIPTION

The `NewMenu` function creates a menu with the specified title, assigns it the specified menu ID, creates a menu record for the menu, and returns a handle to the menu record. It sets up the menu record to use the standard menu definition procedure (and it reads the standard menu definition procedure into memory if it isn't already there). The `NewMenu` function does not insert the newly created menu into the current menu list.

After creating a menu with `NewMenu`, use `AppendMenu`, `InsertMenuItem`, `AppendResMenu`, or `InsertResMenu` to add menu items to the menu. To add a menu created by `NewMenu` to the current menu list, use the `InsertMenu` procedure. To update the menu bar with any new menu titles, use the `DrawMenuBar` procedure.

SPECIAL CONSIDERATIONS

To release the memory associated with a menu that you created using `NewMenu`, first call `DeleteMenu` to remove the menu from the current menu list and to remove any entries for this menu in your application's menu color information table; then call `DisposeMenu` to dispose of the menu's menu record. After disposing of a menu, use `DrawMenuBar` to update the menu bar.

If the `NewMenu` function is unable to create the menu record, it returns `NIL` as its function result.

SEE ALSO

For information on how to add items to a menu, see the description of `AppendMenu` on page 3-124, `InsertMenuItem` on page 3-126, `AppendResMenu` on page 3-128, and `InsertResMenu` on page 3-129. For information on `InsertMenu`, see page 3-108. To dispose of a menu, see the description of `DeleteMenu` on page 3-109 and `DisposeMenu` on page 3-140.

GetMenu

Use the `GetMenu` function to create a menu with the title, items, and other characteristics defined in a `'MENU'` resource with the specified resource ID. You typically use this function only when you create submenus; you can create all your pull-down menus at once using the `GetNewMBar` function, and you can create pop-up menus using the standard pop-up control definition function.

```
FUNCTION GetMenu (resourceID: Integer): MenuHandle;
```

resourceID The resource ID of the `'MENU'` resource that defines the characteristics of the menu. (You usually use the same number for a menu's resource ID as the number that you specify for the menu ID in the menu resource.)

DESCRIPTION

The `GetMenu` function creates a menu according to the specified menu resource, and it also creates a menu record for the menu. It reads the menu definition procedure (specified in the menu resource) into memory if it isn't already in memory, and it stores a handle to the menu definition procedure in the menu record. The `GetMenu` function does not insert the newly created menu into the current menu list.

After reading the `'MENU'` resource, the `GetMenu` function searches for an `'mctb'` resource with the same resource ID as the `'MENU'` resource. If `GetMenu` finds this `'mctb'` resource, it uses the information in the `'mctb'` resource to add entries for this menu to the application's menu color information table. The `GetMenu` function uses `SetMCEntries` to add the entries defined by the `'mctb'` resource to the application's menu color information table. If `GetMenu` doesn't find this `'mctb'` resource, it uses the default colors specified in the menu bar entry of the application's menu color information, or, if the menu bar entry doesn't exist, it uses the standard colors for the menu.

The `GetMenu` function returns a handle to the menu record of the menu. You can use the returned menu handle to refer to this menu in most Menu Manager routines. If `GetMenu` is unable to read the menu or menu definition procedure from the resource file, `GetMenu` returns `NIL`.

After creating a menu with `GetMenu`, you can use `AppendMenu`, `InsertMenuItem`, `AppendResMenu`, or `InsertResMenu` to add more menu items to the menu if necessary.

To add a menu created by `GetMenu` to a menu list, use the `InsertMenu` procedure. To update the menu bar with any new menu titles, use the `DrawMenuBar` procedure.

Storing the definitions of your menus in resources (especially menu titles and menu items) makes your application easier to localize.

▲ **WARNING**
Menus in a resource must not be purgeable. ▲

SPECIAL CONSIDERATIONS

To release the memory associated with a menu that you read from a resource file using `GetMenu`, first call `DeleteMenu` to remove the menu from the menu list and to remove any menu title entry or menu item entries for this menu in the application's menu color information table, then call the Resource Manager procedure `ReleaseResource` to dispose of the menu's menu record. Use `DrawMenuBar` to update the menu bar.

▲ **WARNING**
Call `GetMenu` only once for a particular menu. If you need the handle of a menu currently in the menu list, use `GetMenuHandle` or the Resource Manager function `GetResource`. ▲

For a description of the 'MENU' resource, see "The Menu Resource" on page 3-151; for a sample 'MENU' resource in Rez format, see Listing 3-2 on page 3-48. For information on the 'mctb' resource, see "The Menu Color Information Table Resource" on page 3-155.

For details on how to add items to a menu, see the description of AppendMenu on page 3-124, InsertMenuItem on page 3-126, AppendResMenu on page 3-128, and InsertResMenu on page 3-129. To remove a menu, see the description of DeleteMenu on page 3-109. To update the menu bar, use the DrawMenuBar procedure, described on page 3-113.

Adding Menus to and Removing Menus From the Current Menu List

After creating a menu with NewMenu or GetMenu, use the InsertMenu procedure to insert the menu into the current menu list. Use the DeleteMenu procedure to delete a menu from the current menu list; use the ClearMenuBar procedure to remove all menus from the current menu list.

InsertMenu

Use the InsertMenu procedure to insert an existing menu into the current menu list.

```
PROCEDURE InsertMenu (theMenu: MenuHandle; beforeID: Integer);
```

theMenu   A handle to the menu record of the menu. The NewMenu and GetMenu functions return a handle to a menu record that you can use in this parameter.

beforeID   A number that indicates where in the current menu list the menu should be inserted. InsertMenu inserts the menu into the current menu list before the menu whose menu ID equals the number specified in the beforeID parameter. If the number in the beforeID parameter is 0 (or it isn't the ID of any menu in the menu list), InsertMenu adds the new menu after all others (except before the Help, Keyboard, and Application menus). If the menu is already in the current menu list or the menu list is already full, InsertMenu does nothing.

You can specify –1 for the beforeID parameter to insert a submenu into the current menu list. The submenus in the submenu portion of the menu list do not have to be currently associated with a hierarchical menu item; you can store submenus in the menu list and later specify that a menu item has a submenu if needed. However, note that the MenuKey function scans all menus in the menu list for keyboard equivalents, including submenus that are not associated with any menu item. You should not define keyboard equivalents for submenus that are in the current menu list but not associated with a menu item.

You can also specify –1 for the `beforeID` parameter to insert a pop-up menu into the current menu list. However, if you use the standard pop-up control definition function, the pop-up control automatically inserts the menu into the current menu list according to the needs of the pop-up control.

DESCRIPTION

The `InsertMenu` procedure inserts into the current menu list the menu identified by the specified handle to a menu record. To update the menu bar to reflect the new menu, use `DrawMenuBar`.

SEE ALSO

For details on how to update your application's menu bar, see the description of `DrawMenuBar` on page 3-113.

DeleteMenu

Use the `DeleteMenu` procedure to delete an existing menu from the current menu list.

```
PROCEDURE DeleteMenu (menuID: Integer);
```

menuID   The menu ID of the menu to delete from the current menu list. If the menu list does not contain a menu with the specified menu ID, `DeleteMenu` does nothing.

DESCRIPTION

The `DeleteMenu` procedure deletes the menu identified by the specified menu ID from the current menu list, and it removes all color entries for that menu from the application's menu color information table. `DeleteMenu` does not release the memory occupied by the menu's menu record. To release the memory occupied by the menu's associated data structures, use `DisposeMenu` if you created the menu using `NewMenu`; use the Resource Manager procedure `ReleaseResource` if you created the menu using `GetMenu` or you read the resource in using `GetNewMBar`.

The `DeleteMenu` procedure first checks the submenu portion of the current menu list for a menu ID with the specified ID. If it finds such a menu, it deletes that menu and returns. If `DeleteMenu` doesn't find the menu in the submenu portion, it checks the regular portion of the current menu list. This allows a desk accessory to delete a submenu without deleting an application's menu whose menu ID might conflict with the menu ID defined by a desk accessory.

After deleting a menu, use `DrawMenuBar` to update the menu bar to reflect the changes to the current menu list.

SEE ALSO

For details on how to dispose of a menu's associated data structures using `DisposeMenu`, see "Disposing of Menus" on page 3-140. For information on the `ReleaseResource` procedure, see the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.

ClearMenuBar

Use the `ClearMenuBar` procedure to delete all menus from the current menu list.

```
PROCEDURE ClearMenuBar;
```

DESCRIPTION

The `ClearMenuBar` procedure deletes all menus from the current menu list and deletes all color entries from the application's menu color information table. `ClearMenuBar` does not release the memory occupied by any of the menus' menu records or the menu color information table. To release the memory occupied by the data structures associated with the menus, use `DisposeMenu` for each menu you created using `NewMenu`; use `ReleaseResource` for each menu you created using `GetMenu` or if you read the resource in using `GetNewMBar`.

After deleting all menus from the current menu list, use `DrawMenuBar` to update the appearance of the menu bar.

SEE ALSO

To update your application's menu bar, see the description of `DrawMenuBar` on page 3-113. For information on the `ReleaseResource` procedure, see the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.

Getting a Menu Bar Description From an 'MBAR' Resource

You usually create your application's menu bar by doing the following:

- defining the order and resource ID of your menus in an `'MBAR'` resource

- defining the menus in `'MENU'` resources

- reading in these descriptions using the `GetNewMBar` function

- setting the current menu list to the menu list returned by `GetNewMBar`

- updating the menu bar using `DrawMenuBar`

GetNewMBar

Use the `GetNewMBar` function to read in the definition of a menu bar from an `'MBAR'` resource.

```
FUNCTION GetNewMBar (menuBarID: Integer): Handle;
```

menuBarID   The resource ID of an `'MBAR'` resource that specifies the menus for a menu bar.

DESCRIPTION

The `GetNewMBar` function reads in the definition of a menu bar and its associated menus from an `'MBAR'` resource. The `'MBAR'` resource identifies the order of menus contained in its menu bar. For each menu, it also specifies the menu's resource ID. The `GetNewMBar` function reads in each menu from the `'MENU'` resource with the resource ID specified in the `'MBAR'` resource.

The `GetNewMBar` function creates a menu list for the menu bar defined by the `'MBAR'` resource and returns a handle to the menu list. (If the resource isn't already in memory, `GetNewMBar` reads it into memory.) If `GetNewMBar` can't read the resource, `GetNewMBar` returns `NIL`. `GetNewMBar` uses `GetMenu` to read in each individual menu.

After reading in menus from an `'MBAR'` resource, use `SetMenuBar` to make the menu list created by `GetNewMBar` the current menu list. Then use `DrawMenuBar` to update the menu bar.

To release the memory occupied by the data structures associated with the menus in a menu list, use `DisposeMenu` for each menu you created using `NewMenu`; use the Resource Manager procedure `ReleaseResource` for each menu you created using `GetMenu` or if you read the resource in using `GetNewMBar`. To release the memory occupied by a menu list, use the Memory Manager procedure `DisposeHandle`.

SPECIAL CONSIDERATIONS

The `GetNewMBar` function first saves the current menu list and then clears the current menu list and your application's menu color information table. It then creates a new menu list. Before returning a handle to the new menu list, the `GetNewMBar` function restores the current menu list to the previously saved menu list, but `GetNewMBar` does not restore the previous menu color information table. To save and then restore your application's current menu color information table, call the `GetMCInfo` function before `GetNewMBar` and call the `SetMCInfo` procedure afterward.

While you supply only the resource ID of an `'MBAR'` resource to the `GetNewMBar` function, your application often needs to use the menu IDs defined in each of your menus' `'MENU'` resources. Most Menu Manager routines require either a menu ID or a handle to a menu record to perform operations on a specific menu. For menus in the current menu list, you can use the `GetMenuHandle` function to get the handle to a menu record of a menu with a given menu ID.

For a description of the `'MENU'` resource, see "The Menu Resource" on page 3-151; for a sample `'MENU'` resource in Rez format, see Listing 3-2 on page 3-48. For a description of the `'MBAR'` resource, see "The Menu Bar Resource" on page 3-155; for a sample `'MBAR'` resource in Rez format, see Listing 3-4 on page 3-49. For information on the `'mctb'` resource, see "The Menu Color Information Table Resource" on page 3-155. For information about the Resource Manager, see *Inside Macintosh: More Macintosh Toolbox.*

Getting and Setting the Menu Bar

You can use the `GetMenuBar` function to get a handle to a copy of the current menu list. Use the `SetMenuBar` procedure to set the current menu bar to a menu list previously returned by `GetMenuBar` or `GetNewMBar`. You can get the height of the menu bar using the `GetMBarHeight` function.

GetMenuBar

Use the `GetMenuBar` function to get a handle to a copy of the current menu list.

```
FUNCTION GetMenuBar: Handle;
```

DESCRIPTION

The `GetMenuBar` function creates a copy of the current menu list and returns a handle to the copy. You can save the returned menu list and then add menus to or remove menus from the current menu list (using `InsertMenu`, `DeleteMenu`, or `ClearMenuBar`). You can later restore the saved menu list using `SetMenuBar`.

To release the memory occupied by a saved menu list, use the Memory Manager's `DisposeHandle` procedure.

▲ **WARNING**
`GetMenuBar` doesn't copy the menu records, just the menu list (which contains handles to the menu records). Do not dispose of any menus in a saved menu list if you wish to restore the menu list later. ▲

SetMenuBar

Use the `SetMenuBar` procedure to set the current menu list to a specified menu list.

```
PROCEDURE SetMenuBar (menuList: Handle);
```

menuList   A handle to a menu list that specifies the menus for a menu bar. You should specify a handle returned by `GetMenuBar` or `GetNewMBar`.

DESCRIPTION

The `SetMenuBar` procedure copies the given menu list to the current menu list. As with `GetMenuBar`, `SetMenuBar` doesn't copy the menu records, just the menu list (which contains handles to the menu records).

You can use `SetMenuBar` to restore a menu list that you previously saved using `GetMenuBar` or to set the current menu list to a menu list created by `GetNewMBar`.

The `SetMenuBar` procedure sets only the current menu list; to update the menu bar according to the new menu list, use the `DrawMenuBar` procedure.

GetMBarHeight

Use the `GetMBarHeight` function if you need to determine the current height of the menu bar. When the Roman script system is the current system script, the menu bar is 20 pixels high. If a non-Roman script is the current system script, the menu bar may be greater than 20 pixels high to accommodate the current system font.

```
FUNCTION GetMBarHeight: Integer;
```

DESCRIPTION

The `GetMBarHeight` function returns the current height, in pixels, of the menu bar.

Drawing the Menu Bar

Whenever your application adds menus to or removes menus from the current menu list, you should update the titles of the menus in the menu bar using the `DrawMenuBar` procedure. If you change the enabled state of a menu, you should call `DrawMenuBar` to update the menu title accordingly. Alternatively, you can use the `InvalMenuBar` procedure instead of `DrawMenuBar` to invalidate the menu bar; this causes the Event Manager to redraw the menu bar as part of its normal processing of update events.

DrawMenuBar

Use the `DrawMenuBar` procedure to draw the menu bar based on the current menu list.

```
PROCEDURE DrawMenuBar;
```

DESCRIPTION

The `DrawMenuBar` procedure draws (or redraws) the menu bar according to the current menu list. You must call `DrawMenuBar` to update the menu bar after adding menus to or deleting menus from the current menu list using `InsertMenu` or `DeleteMenu`, after setting the current menu list using `SetMenuBar`, after changing the enabled state of a menu, or after any other routine that changes the current menu list.

InvalMenuBar

Use the `InvalMenuBar` procedure to invalidate the menu bar.

```
PROCEDURE InvalMenuBar;
```

The `InvalMenuBar` procedure marks the menu bar as changed and in need of updating. When the Event Manager scans update regions for regions that require updating, the Event Manager also checks to determine whether the menu bar requires updating (because of a call to `InvalMenuBar`). If the menu bar needs updating, the Event Manager calls the `DrawMenuBar` procedure to draw the menu bar.

You can use `InvalMenuBar` instead of `DrawMenuBar` to minimize blinking in the menu bar. For example, if you have several application-defined routines that can change the enabled state of a menu and each calls `DrawMenuBar`, you can replace the calls to `DrawMenuBar` with calls to `InvalMenuBar`. In this way the menu bar is redrawn only once instead of multiple times in quick succession. If you need to make immediate changes to the menu bar, use `DrawMenuBar`. If you want to redraw the menu bar at most once each time through your event loop, use `InvalMenuBar`. The `InvalMenuBar` procedure is available only in System 7.

Responding to the User's Choice of a Menu Command

When the user presses the mouse button while the cursor is in the menu bar, your application should call the `MenuSelect` function to allow the user to choose a command from the menu bar. If the user presses the mouse button while the cursor is over a pop-up menu that does not use the standard pop-up control definition function, your application should call the `PopUpMenuSelect` function to allow the user to make a choice from the pop-up menu.

You should also allow the user to choose a menu command by typing a keyboard equivalent. When the user presses a key on the keyboard, your application should determine if the Command key was pressed at the same time, and, if so, your application should call the `MenuKey` function to map this keyboard combination to any corresponding Command-key equivalent.

If the user chooses an item, both the `MenuSelect` and `MenuKey` functions highlight the title of the menu containing the chosen item and report the user's choice to your application. Your application should perform the corresponding command and, when finished, should unhighlight the menu title using the `HiliteMenu` procedure to indicate to the user that the command is completed.

If the user releases the mouse button while the cursor is over a disabled item or types the keyboard equivalent of a disabled item, `MenuSelect` and `MenuKey` do not report the menu ID or menu item of the item. To determine if the user chose a disabled item (for example, so that your application can provide assistance to the user or explain to the user why the command is disabled), you can use the `MenuChoice` function to return the menu ID and menu item of the disabled menu command.

Your application should adjust its menus before calling `MenuSelect` or `MenuKey`. For example, you should enable or disable menu items as appropriate and add any applicable checkmarks or dashes to items that show attributes.

MenuSelect

Use the `MenuSelect` function to allow the user to choose a menu item from the menus in your application's menu bar.

```
FUNCTION MenuSelect (startPt: Point): LongInt;
```

startPt   The point (in global coordinates) representing the location of the cursor at the time the mouse button was pressed.

DESCRIPTION

When the user presses the mouse button while the cursor is in the menu bar, your application receives a mouse-down event. To handle mouse-down events in the menu bar, pass the location of the cursor at the time of the mouse-down event as the `startPt` parameter to `MenuSelect`. The `MenuSelect` function displays and removes menus as the user moves the cursor over menu titles in the menu bar, and it handles all user interaction until the user releases the mouse button.

As the user drags the cursor through the menu bar, the `MenuSelect` function highlights the title of the menu the cursor is currently over and displays all items in that menu. If the user moves the cursor so that it is over a different menu, the `MenuSelect` function removes the previous menu and unhighlights its menu title.

The `MenuSelect` function highlights and unhighlights menu items as the user drags the cursor over the items in a menu. The `MenuSelect` function highlights a menu item if the item is enabled and the cursor is currently over it; it removes such highlighting when the user moves the cursor to another menu item. The `MenuSelect` function does not highlight disabled menu items.

If the user chooses an enabled menu item (including any item from a submenu), the `MenuSelect` function returns a value as its function result that indicates which menu and menu item the user chose. The high-order word of the function result contains the menu ID of the menu, and the low-order word contains the item number of the menu item chosen by the user. The `MenuSelect` function leaves the menu title highlighted; after performing the chosen task your application should unhighlight the menu title using the `HiliteMenu` procedure.

If the user chooses an item from a submenu, `MenuSelect` returns the menu ID of the submenu in the high-order word and the item chosen by the user in the low-order word of its function result. The `MenuSelect` function also highlights the title of the menu in the menu bar that the user originally displayed in order to begin traversing to the submenu. After performing the chosen task, your application should unhighlight the menu title.

Menu Manager

If the user releases the mouse button while the cursor is over a disabled item, in the menu bar, or outside of any menu, the MenuSelect function returns 0 in the high-order word of its function result and the low-order word is undefined. If it is necessary for your application to find the item number of the disabled item, your application can call MenuChoice to return the menu ID and menu item.

If the user chooses an enabled item in a menu that a desk accessory has inserted into your application's menu list, MenuSelect uses the SystemMenu procedure to process this occurrence and returns 0 to your application in the high-order word.

SPECIAL CONSIDERATIONS

When the MenuSelect function pulls down a menu, it stores the bits behind the menu as a relocatable object in the application heap of your application.

ASSEMBLY-LANGUAGE INFORMATION

The InitMenus and InitProcMenu procedures initialize the MenuHook and MBarHook global variables to 0. If you choose, you can store the addresses of routines that MenuSelect calls in these global variables. The MenuHook global variable contains the address (if any) of a routine that MenuSelect calls repeatedly while the mouse button is down. MenuSelect does not pass any parameters to this routine.

The MBarHook global variable contains the address (if any) of a routine that MenuSelect calls after a menu title is highlighted and the menu rectangle is calculated but before the menu is drawn. The menu rectangle is the rectangle (in global coordinates) in which the menu will be drawn. MenuSelect passes a pointer to the menu rectangle on the stack. If you provide the address of a routine in the MBarHook global variable, it should normally return 0 in the D0 register, indicating that MenuSelect should continue; returning 1 causes MenuSelect to cancel its operation and return immediately to the application.

The MenuSelect function uses the global variable MBarEnable to determine if all menus in the current menu bar belong to a desk accessory or an application. If the MBarEnable global variable is nonzero, then all menus in the current menu bar belong to a desk accessory. If the MBarEnable global variable is 0, then all menus in the current menu bar belong to an application. If you're writing a desk accessory, you may need to set the MBarEnable global variable to a nonzero value; if you're writing an application, you should not change the value of the MBarEnable global variable.

The global variable TheMenu contains the ID of the currently highlighted menu in the menu bar. If the user chooses an item from a submenu, TheMenu contains the menu ID of the submenu, not the menu to which the submenu is attached.

SEE ALSO

For information on adjusting your application's menus before calling MenuSelect, see "Adjusting the Menus of an Application" beginning on page 3-73.

See the description of the `HiliteMenu` procedure on page 3-119 for details on how to unhighlight a menu. For information on how to determine if the user chose a disabled item, see the description of the `MenuChoice` function on page 3-118.

MenuKey

If the user presses another key while holding down the Command key, call the `MenuKey` function to determine if the keyboard combination maps to the keyboard equivalent of a menu item in a menu in the current menu list.

```
FUNCTION MenuKey (ch: Char): LongInt;
```

ch   The 1-byte character representing the key pressed by the user in combination with the Command key.

DESCRIPTION

The `MenuKey` function maps the given character to the menu and menu item with that keyboard equivalent. The `MenuKey` function returns as its function result a value that indicates the menu ID and menu item that has the keyboard equivalent corresponding to the given character.

The `MenuKey` function does not distinguish between uppercase and lowercase letters. It takes the 1-byte character passed to it and calls the `UpperText` procedure (which provides localizable uppercase conversion of the character). Thus, `MenuKey` translates any lowercase character to uppercase when comparing a keyboard event to keyboard equivalents. This allows a user to invoke a keyboard equivalent command, such as the Copy command, by pressing the Command key and "c" or "C". For consistency between applications, you should define the keyboard equivalents of your commands so that they appear in uppercase in your menus.

If the given character maps to an enabled menu item in the current menu list, `MenuKey` highlights the menu title of the chosen menu, returns the menu ID in the high-order word of its function result, and returns the chosen menu item in the low-order word of its function result. After performing the chosen task, your application should unhighlight the menu title using the `HiliteMenu` procedure.

If the given character does not map to an enabled menu item in the current menu list, `MenuKey` returns 0 in its high-order word and the low-order word is undefined.

If the given character maps to a menu item in a menu that a desk accessory has inserted into your application's menu list, `MenuSelect` uses the `SystemMenu` procedure to process this occurrence and returns 0 to your application in the high-order word.

You should not define menu items with identical keyboard equivalents. The `MenuKey` function scans the menus from right to left and the items from top to bottom. If you have defined more than one menu item with identical keyboard equivalents, `MenuKey` returns the first one it finds.

The `MenuKey` function first searches the regular portion of the current menu list for a menu item with a keyboard equivalent matching the given key. If it doesn't find one there, it searches the submenu portion of the current menu list. If the given key maps to a menu item in a submenu, `MenuKey` highlights the menu title in the menu bar that the user would normally pull down to begin traversing to the submenu. Your application should perform the desired command and then unhighlight the menu title.

You shouldn't assign a Command–Shift–number key sequence to a menu item as its keyboard equivalent; Command–Shift–number key sequences are reserved for use as `'FKEY'` resources. Command–Shift–number key sequences are not returned to your application, but instead are processed by the Event Manager. The Event Manager invokes the `'FKEY'` resource with a resource ID that corresponds to the number that activates it.

Apple reserves the Command-key codes $1B (Control-[ ) through $1F (Control-_) to indicate meanings other than keyboard equivalents. `MenuKey` ignores these character codes and returns a function result of 0 if you specify any of these values in the `ch` parameter. Your application should not use these character codes for its own use.

The global variable `TheMenu` contains the ID of the currently highlighted menu in the menu bar. If the user chooses an item from a submenu, `TheMenu` contains the menu ID of the submenu, not the menu to which the submenu is attached.

▲ **WARNING**
Do not define a "circular" hierarchical menu—that is, a hierarchical menu in which a submenu has a submenu whose submenu is a hierarchical menu higher in the chain. If `MenuKey` detects a circular hierarchical menu, it creates a system error with error number 86. ▲

SEE ALSO

To unhighlight a menu, use the `HiliteMenu` procedure, described on page 3-119. To provide support for keyboard equivalents other than Command-key equivalents, see the discussion of `'KCHR'` resources in *Inside Macintosh: Text*.

MenuChoice

If your application needs to find the item number of a disabled menu item that the user attempted to choose, you can use the `MenuChoice` function to return the chosen menu item.

```
FUNCTION MenuChoice: LongInt;
```

DESCRIPTION

If the user chooses a disabled menu item, the `MenuChoice` function returns a value that indicates which menu and menu item the user chose. The high-order word of the

function result contains the menu ID of the menu, and the low-order word contains the item number of the menu item chosen by the user.

The `MenuChoice` function returns 0 as the low-order word of its function result if the mouse button was released while the cursor was in the menu bar or outside the menu.

SPECIAL CONSIDERATIONS

The Menu Manager updates the global variable `MenuDisable` whenever a menu is displayed. As the user moves the cursor over each item, the Menu Manager calls the menu definition procedure of the menu to update the `MenuDisable` global variable to reflect the current menu ID and menu item. The standard menu definition procedure updates the global variable `MenuDisable` appropriately. If your application uses its own menu definition procedure, your menu definition procedure should support this feature; if you use a menu definition procedure that does not update the global variable `MenuDisable` appropriately, the result returned by `MenuChoice` is undefined.

HiliteMenu

You can use the `HiliteMenu` procedure to highlight or unhighlight menu titles. For example, after performing a menu command chosen by the user, use the `HiliteMenu` procedure to unhighlight the menu title.

```
PROCEDURE HiliteMenu (menuID: Integer);
```

menuID   The menu ID of the menu whose title should be highlighted. If the menu title of the specified menu is already highlighted, `HiliteMenu` does nothing. If the menu ID is 0 or the specified menu ID isn't in the current menu list, `HiliteMenu` unhighlights whichever menu title is currently highlighted (if any).

DESCRIPTION

The `MenuSelect` and `MenuKey` functions highlight the title of the menu containing the item chosen by the user. After performing the chosen task, your application should unhighlight the menu title by calling `HiliteMenu` and passing 0 in the `menuID` parameter.

The `HiliteMenu` procedure highlights a menu title by first saving the bits behind the title rectangle and then drawing the highlighted title. `HiliteMenu` unhighlights a menu title by restoring the bits behind the menu title.

The global variable `TheMenu` contains the ID of the currently highlighted menu in the menu bar. If the user chooses an item from a submenu, `TheMenu` contains the menu ID of the submenu, not the menu to which the submenu is attached.

PopUpMenuSelect

To display a pop-up menu without using the standard pop-up control definition function, use the `PopUpMenuSelect` function to display the pop-up menu anywhere on the screen. If your application uses the standard pop-up control definition function, your application does not need to use `PopUpMenuSelect`.

```
FUNCTION PopUpMenuSelect (menu: MenuHandle;
                            Top: Integer; Left: Integer;
                            PopUpItem: Integer)
                            : LongInt;
```

menu   A handle to the menu record of the menu. The `NewMenu`, `GetMenu`, and `GetMenuHandle` functions return a handle to a specified menu's menu record.

Top   The top coordinate of the pop-up box when it is closed. This value should be in global coordinates.

Left   The left coordinate of the pop-up box when it is closed. This value should be in global coordinates.

PopUpItem   The item number of the current item minus 1. This value should correspond to the user's previous choice from this menu. If the user has not previously made a choice, this value should be set to the default value.

DESCRIPTION

The `PopUpMenuSelect` function uses the location specified by the `Top` and `Left` parameters to determine where to display the specified item of the pop-up menu. The `PopUpMenuSelect` function displays the pop-up menu so that the menu item specified in the `PopUpItem` parameter appears highlighted at the specified location. Figure 3-24 on page 3-34 shows the pop-up title and pop-up box of a pop-up menu.

The `PopUpMenuSelect` function highlights and unhighlights menu items and handles all user interaction until the user releases the mouse button. The `PopUpMenuSelect` function returns the menu ID of the chosen menu in the high-order word of its function result and the chosen menu item in the low-order word.

Your application is responsible for highlighting the pop-up title, setting the mark of the current menu item appropriately, and drawing the text and downward-pointing indicator in the pop-up box before calling `PopUpMenuSelect`. Your application should also make sure the pop-up menu is in the submenu portion of the current menu list before calling `PopUpMenuSelect`. (You can use the `InsertMenu` procedure and specify –1 in the `beforeID` parameter to insert the pop-up menu into the current menu list.)

After calling PopUpMenuSelect, your application can delete the pop-up menu from the current menu list or leave it in the current menu list.

Your application is also responsible for storing the current value of the menu item, drawing the text and downward-pointing indicator in the pop-up box, and unhighlighting the pop-up title after calling PopUpMenuSelect. If you use the standard pop-up control definition function, these actions are performed for you by the pop-up control and your application does not need to call PopUpMenuSelect.

When implementing pop-up menus, you should follow the guidelines for pop-up menus described in *Macintosh Human Interface Guidelines.* For example, you should define the pop-up box of your pop-up menu as a rectangle that is the same height as a menu item, with a one-pixel drop shadow, and should make the pop-up box wide enough to show the currently selected item and a downward-pointing indicator.

SystemMenu

The MenuSelect and MenuKey functions call the SystemMenu procedure when the user chooses an item in a menu that belongs to a desk accessory launched in your application's partition. Your application should not need to call the SystemMenu procedure.

```
PROCEDURE SystemMenu (menuResult: LongInt);
```

menuResult   The value that indicates the menu and menu item chosen by the user. The menu ID is in the high-order word, and the menu item is in the low-order word. The menu ID for a menu belonging to a desk accessory is a negative number.

DESCRIPTION

The SystemMenu procedure directs the desk accessory to perform the appropriate action for the given menu item by calling the desk accessory's control routine and passing the accMenu constant in the csCode parameter. The desk accessory should perform the desired action and return. See *Inside Macintosh: Devices* for more information on desk accessories.

ASSEMBLY-LANGUAGE INFORMATION

If you're writing a desk accessory, you may need to set the MBarEnable global variable to appropriate values. If the MBarEnable global variable is nonzero, then all menus in the current menu bar belong to a desk accessory. If the MBarEnable global variable is 0, then all menus in the current menu bar belong to an application. If you're writing an application, you should not change the value of the MBarEnable global variable.

SystemEdit

When the user chooses one of the standard editing commands in the Edit menu (Undo, Cut, Copy, Paste, and Clear), call the `SystemEdit` function to determine whether the active window belongs to a desk accessory that is launched in your application's partition. If so, the `SystemEdit` function directs the desk accessory to perform the editing command and returns `TRUE`. If the active window does not belong to a desk accessory launched in your application's partition, `SystemEdit` returns `FALSE` and your application should process the command.

```
FUNCTION SystemEdit (editCmd: Integer): Boolean;
```

editCmd   The item number of the standard editing command chosen by the user.

Getting a Handle to a Menu Record

Most Menu Manager routines that manage menus require that you specify a handle to the menu record of the menu on which you want to perform an operation. You can use the `HMGetHelpMenuHandle` function to get a handle to your application's Help menu. Use the `GetMenuHandle` function to get a handle to the menu record of any of your application's other pull-down menus or submenus in the current menu list. For pop-up menus that use the standard control definition function, you can access the control record to get the menu's handle.

GetMenuHandle

You can use the `GetMenuHandle` function to get a handle to the menu record of any of your application's menus other than its Help menu. (Use the `HMGetHelpMenuHandle` function to get a handle to the menu record of your application's Help menu.) The `GetMenuHandle` function is also available as the `GetMHandle` function.

```
FUNCTION GetMenuHandle (menuID: Integer): MenuHandle;
```

menuID   The menu ID of the menu. (Note that this is not the resource ID, although you often assign the menu ID so that it matches the resource ID.) You assign a menu ID in the `'MENU'` resource of a menu. If you do not define your menus in `'MENU'` resources, you can assign a menu ID using `NewMenu`.

DESCRIPTION

The `GetMenuHandle` function returns a handle to the menu record of the menu having the specified menu ID. If the menu is in the current menu list, `GetMenuHandle` returns a handle to the menu record of the menu as its function result. Otherwise, `GetMenuHandle` returns `NIL` as its function result.

SPECIAL CONSIDERATIONS

To get a handle to a menu record of a pop-up menu that you create using the pop-up control definition function, dereference the cntrlData field of the pop-up menu's control record instead of using GetMenuHandle.

HMGetHelpMenuHandle

Use the HMGetHelpMenuHandle function to get a handle to the menu record of your application's Help menu.

```
FUNCTION HMGetHelpMenuHandle (VAR mh: MenuHandle): OSErr;
```

mh   The HMGetHelpMenuHandle function returns a copy of a handle to your application's Help menu in this parameter.

DESCRIPTION

The HMGetHelpMenuHandle function returns in the mh parameter a copy of a handle to the menu record of your application's Help menu. With this handle, you can append items to your application's Help menu by using the AppendMenu procedure or other related Menu Manager routines. The Help Manager automatically adds the divider that separates your items from the rest of the Help menu items.

Be sure to define help balloons for your items in the Help menu by creating an 'hmnu' resource and specifying the kHMHelpMenuID constant as its resource ID.

The Menu Manager functions MenuSelect and MenuKey return a result with the menu ID in the high-order word and the menu item in the low-order word. The MenuSelect function (and the MenuKey function, if the user chooses an item with a keyboard equivalent) returns the kHMHelpMenuID constant in the high-order word when the user chooses an appended item from the Help menu. The menu item number of the appended menu item is returned in the low-order word of the function result. Apple reserves the right to change the number of standard items in the Help menu. To determine the number of items in the Help menu, call the CountMItems function.

SPECIAL CONSIDERATIONS

Do not use the GetMenuHandle function to get a handle to the menu record of the Help menu. GetMenuHandle returns a handle to the menu record of the global Help menu, not the menu record of the Help menu that is specific to your application.

RESULT CODES

noErr	0	No error
paramErr	–50	Error in parameter list
memFullErr	–108	Not enough room in heap zone
resNotFound	–192	Unable to read resource
hmHelpManagerNotInited	–855	Help menu not set up

For examples of how to add items to your application's Help menu and how to handle the user's choice of an item in the Help menu, see Listing 3-14 on page 3-68 and Listing 3-26 on page 3-81. See the chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox* for information on creating help balloons for the menus of your application.

Adding and Deleting Menu Items

You can add the names of all resources of a specified type to a menu using the `InsertResMenu` or `AppendResMenu` procedure. You can add menu items that you define to a menu using the `AppendMenu` or `InsertMenuItem` procedure. You can also delete menu items using the `DeleteMenuItem` procedure. In most cases you should not insert or delete individual menu items from an already existing menu unless the user expects a menu (such as a list of currently open documents) to change.

If you add menu items using the `AppendMenu` or `InsertMenuItem` procedure, you should define in resources the text and other characteristics of the menu items that you add. This makes your application easier to localize for other regions.

AppendMenu

Use the `AppendMenu` procedure to append one or more items to a menu previously created using `NewMenu`, `GetMenu`, or `GetNewMBar`.

```
PROCEDURE AppendMenu (menu: MenuHandle; data: Str255);
```

menu   A handle to the menu record of the menu to which you wish to append the menu item or items.

data   A string that defines the characteristics of the new menu item or items. Note that in most cases you should store the text of a menu item in a resource, so that your menu items can be more easily localized. The `AppendMenu` procedure appends the menu items in the order in which they are listed in the `data` parameter.

DESCRIPTION

The `AppendMenu` procedure appends any defined menu items to the specified menu. The menu items are added to the end of the menu. You specify the text of any menu items and their characteristics in the `data` parameter. You can embed metacharacters in the string to define various characteristics of a menu item.

Here are the metacharacters that you can specify in the `data` parameter:

Metacharacter	Description
; or Return	Separates menu items.
^	When followed by an icon number, defines the icon for the item. If the keyboard equivalent field contains $1C, this number is interpreted as a script code.
!	When followed by a character, defines the mark for the item. If the keyboard equivalent field contains $1B, this value is interpreted as the menu ID of a submenu of this menu item.
<	When followed by one or more of the characters B, I, U, O, and S, defines the character style of the item to Bold, Italic, Underline, Outline, or Shadow, respectively.
/	When followed by a character, defines the keyboard equivalent for the item. When followed by $1B, specifies that this menu item has a submenu. To specify that the menu item has a script code, small icon, or reduced icon, use the `SetItemCmd` procedure to set the keyboard equivalent field to $1C, $1D, or $1E, respectively.
(	Defines the menu item as disabled.

You can specify any, all, or none of these metacharacters in the text string. The metacharacters that you specify aren't displayed in the menu item. (To use any of these metacharacters in the text of a menu item, first use `AppendMenu`, specifying at least one character as the item's text, and then use the `SetMenuItemText` procedure to set the item's text to the desired string.)

Note

If you add menu items using the `AppendMenu` procedure, you should define the text and any marks or keyboard equivalents in resources for easier localization. ◆

You can specify the first character that defines the text of a menu item as a hyphen to create a divider line. The string in the `data` parameter can be blank (containing one or more spaces), but it should not be an empty string.

If you do not define a specific characteristic of a menu item, the `AppendMenu` procedure assigns the default characteristic to the menu item. If you do not define any characteristic other than the text for a menu item, the `AppendMenu` procedure inserts the menu item so that it appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent.

You can use `AppendMenu` to append items to a menu regardless of whether the menu is in the current menu list.

SEE ALSO

See "Adding Items to a Menu" on page 3-64 for examples of appending items to a menu.

InsertMenuItem

Use the `InsertMenuItem` procedure to insert one or more items to a menu previously created using `NewMenu`, `GetMenu`, or `GetNewMBar`.

The `InsertMenuItem` procedure is also available as the `InsMenuItem` procedure.

```
PROCEDURE InsertMenuItem (theMenu: MenuHandle; itemString: Str255;
                          afterItem: Integer);
```

theMenu   A handle to the menu record of the menu to which you wish to add the menu item or items.

itemString
            A string that defines the characteristics of the new menu items. Note that in most cases you should store the text of a menu item in a resource, so that your menu items can be more easily localized. You can specify the contents of the `itemString` parameter using metacharacters; the `InsertMenuItem` procedure accepts the same metacharacters as the `AppendMenu` procedure. However, if you specify multiple items, the `InsertMenuItem` procedure inserts the items in the reverse of their order in the `itemString` parameter.

afterItem   The item number of the menu item after which the new menu items are to be added. Specify 0 in the `afterItem` parameter to insert the new items before the first menu item; specify the item number of a current menu item to insert the new menu items after it; specify a number greater than or equal to the last item in the menu to append the new items to the end of the menu.

DESCRIPTION

The `InsertMenuItem` procedure inserts any defined menu items to the specified menu. The menu items are inserted according to the location specified by the `afterItem` parameter. You specify the text of any menu items and their characteristics in the `itemString` parameter. You can embed metacharacters in the string you specify to define various characteristics of a menu item. The metacharacters aren't displayed in the menu.

Here are the metacharacters you can specify in the `itemString` parameter:

Metacharacter	Description
; or Return	Separates menu items.
^	When followed by an icon number, defines the icon for the item. If the keyboard equivalent field contains $1C, this number is interpreted as a script code.
!	When followed by a character, defines the mark for the item. If the keyboard equivalent field contains $1B, this value is interpreted as the menu ID of a submenu of this menu item.

Metacharacter	Description
<	When followed by one or more of the characters B, I, U, O, and S, defines the character style of the item to Bold, Italic, Underline, Outline, or Shadow, respectively.
/	When followed by a character, defines the keyboard equivalent for the item. When followed by $1B, specifies that this menu item has a submenu. To specify that the menu item has a script code, small icon, or reduced icon, use the SetItemCmd procedure to set the keyboard equivalent field to $1C, $1D, or $1E, respectively.
(	Defines the menu item as disabled.

You can specify any, all, or none of these metacharacters in the text string. The metacharacters that you specify aren't displayed in the menu item. To use any of these metacharacters in the text of a menu item, first use InsertMenuItem, specifying at least one character as the item's text, and then use the SetMenuItemText procedure to set the item's text to the desired string.

Note

If you add menu items using the InsertMenuItem procedure, you should define the text and any marks or keyboard equivalents in resources for easier localization. ◆

You can specify the first character that defines the text of a menu item as a hyphen to create a divider line. The string in the itemString parameter can be blank (containing one or more spaces), but it should not be an empty string.

If you do not define a specific characteristic of a menu item, the InsertMenuItem procedure assigns the default characteristic to the menu item. If you do not define any characteristic other than the text for a menu item, the InsertMenuItem procedure inserts the menu item so that it appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent.

You can use InsertMenuItem to insert items into a menu regardless of whether the menu is in the current menu list.

SEE ALSO

See "Adding Items to a Menu" beginning on page 3-64 for examples.

DeleteMenuItem

Use the DeleteMenuItem procedure to delete an item from a menu. The DeleteMenuItem procedure is also available as the DelMenuItem procedure.

```
PROCEDURE DeleteMenuItem (theMenu: MenuHandle; item: Integer);
```

theMenu   A handle to the menu record of the menu from which you want to delete
                 the menu item.

item   The item number of the menu item to delete. If you specify 0 or a number
                 greater than the last item in the menu, DeleteMenuItem does not delete
                 any item from the menu.

DESCRIPTION

The DeleteMenuItem procedure deletes a specified menu item from a menu. The
DeleteMenuItem procedure also deletes the item's menu item entry from your
application's menu color information table (if an entry exists). You should not delete
items from an existing menu unless the user expects the menu (such as a menu that lists
open documents) to change.

AppendResMenu

Use the AppendResMenu procedure to search all resource files open to your application
for a given resource type and to append the names of any resources it finds to a specified
menu. The specified menu must have been previously created using NewMenu,
GetMenu, or GetNewMBar.

The AppendResMenu procedure is also available as the AddResMenu procedure.

```
PROCEDURE AppendResMenu (theMenu: MenuHandle; theType: ResType);
```

theMenu   A handle to the menu record of the menu to which to append the names
                 of any resources of a given type that AppendResMenu finds.

theType   A four-character code that identifies the resource type for which to search.

DESCRIPTION

The AppendResMenu procedure searches all resource files open to your application for
resources of the type defined by the parameter theType. It appends the names of any
resources it finds of the given type to the end of the specified menu. AppendResMenu
appends the names of found resources in alphabetical order; it does not alphabetize
items already in the menu. The AppendResMenu procedure does not add resources with
names that begin with a period (.) or a percent sign (%) to the menu.

The AppendResMenu procedure assigns default characteristics to each menu item. Each
appended menu item appears in the menu as an enabled item, without an icon or a
mark, in the plain character style, and without a keyboard equivalent. To get the name or
to change other characteristics of an item appended by AppendResMenu, use the Menu
Manager routines described in "Getting and Setting the Appearance of Menu Items"
beginning on page 3-130.

If you specify that `AppendResMenu` add resources of type `'DRVR'` to your Apple menu, `AppendResMenu` adds the name (and icon) of each item in the Apple Menu Items folder to the menu.

If you specify that `AppendResMenu` append resources of type `'FONT'` or `'FOND'`, the Menu Manager performs special processing for any resources it finds that have font numbers greater than $4000. If the script system associated with the font name is installed in the system, `AppendResMenu` stores information in the `itemDefinitions` array (in the `itemIcon` and `itemCmd` fields for that item) in the menu's menu record. This allows the Menu Manager to display the font name in the correct script.

SPECIAL CONSIDERATIONS

The `AppendResMenu` procedure calls the Resource Manager procedure `SetResLoad` (specifying `TRUE` in the `load` parameter) before returning. The `AppendResMenu` procedure reads the resource data of the resources it finds into memory. If your application does not want the Resource Manager to read resource data into memory when your application calls other routines that read resources, you need to call `SetResLoad` and specify `FALSE` in the `load` parameter after `AppendResMenu` returns.

SEE ALSO

Listing 3-15 on page 3-69 shows a sample that adds items from the Apple Menu Items folder to the Apple menu, and Listing 3-16 on page 3-70 shows a sample that adds font names to a menu. See *Inside Macintosh: More Macintosh Toolbox* for information on the Resource Manager.

InsertResMenu

Use the `InsertResMenu` procedure to search all resource files open to your application for a given resource type and to insert the names of any resources it finds to a specified menu. The items are inserted after the specified menu item. The specified menu must have been previously created using `NewMenu`, `GetMenu`, or `GetNewMBar`.

```
PROCEDURE InsertResMenu (theMenu: MenuHandle; theType: ResType;
                         afterItem: Integer);
```

theMenu   A handle to the menu record of the menu to which to add the names of any resources of a given type that `InsertResMenu` finds.

theType   A four-character code that identifies the resource type for which to search.

afterItem   A number that indicates where in the menu to insert the names of any resources of the given type that `InsertResMenu` finds. Specify 0 in the `afterItem` parameter to insert the items before the first menu item; specify the item number of a menu item already in the menu to insert the items after the specified item number. If you specify a number greater than or equal to the last item in the menu, the items are inserted at the end of the menu.

3

Menu Manager

DESCRIPTION

The `InsertResMenu` procedure searches all resource files open to your application for resources of the type defined by the parameter `theType`. It inserts the names of any resources it finds of the given type at the specified location in the specified menu. `InsertResMenu` adds the names of found resources in alphabetical order; it does not alphabetize items already in the menu.

The `InsertResMenu` procedure does not add resources with names that begin with a period (`.`) or a percent sign (`%`) to the menu.

The `InsertResMenu` procedure assigns default characteristics to each menu item. Each appended menu item appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent. To get the name or to change other characteristics of an item appended by `InsertResMenu`, use the Menu Manager routines described in the next section, "Getting and Setting the Appearance of Menu Items."

If you specify that `InsertResMenu` add resources of type `'DRVR'` to your Apple menu, `InsertResMenu` adds the name (and icon) of each item in the Apple Menu Items folder to the menu.

If you specify that `InsertResMenu` add resources of type `'FONT'` or `'FOND'`, the Menu Manager performs special processing for any resources it finds that have font numbers greater than $4000. If the script associated with the font name is currently active, `InsertResMenu` stores information in the `itemDefinitions` array (in the `itemIcon` and `itemCmd` fields for that item) in the menu's menu record that allows the Menu Manager to display the font name in the correct script.

SPECIAL CONSIDERATIONS

The `InsertResMenu` procedure calls the Resource Manager procedure `SetResLoad` (specifying `TRUE` in the `load` parameter) before returning. The `InsertResMenu` procedure reads the resource data of the resources it finds into memory. If your application does not want the Resource Manager to read resource data into memory when your application calls other routines that read resources, you need to call `SetResLoad` and specify `FALSE` in the `load` parameter after `InsertResMenu` returns.

Getting and Setting the Appearance of Menu Items

You can get information about the characteristics of a menu item using Menu Manager routines. For example, you can get an item's text, style, mark, keyboard equivalent, script code, and associated icons. You can also determine if a menu item has a submenu associated with it and the menu ID of the submenu.

You can set the characteristics of a menu item, including associating a submenu with a menu item, using Menu Manager routines. Whenever possible, however, you should define your application's menu items in `'MENU'` resources. This makes your application easier to localize for other regions.

You can also enable and disable menu items or entire menus using Menu Manager routines.

EnableItem

Use the `EnableItem` procedure to enable a menu item or a menu.

```
PROCEDURE EnableItem (theMenu: MenuHandle; item: Integer);
```

theMenu   A handle to the menu record of the menu containing the menu item
           to enable.

item   The item number of the menu item to enable, or 0 to enable the entire
           menu. You cannot individually enable a menu item with an item number
           greater than 31.

           If you specify 0 in the `item` parameter, the `EnableItem` procedure
           enables the menu title and all items in the menu that were not previously
           individually disabled.

DESCRIPTION

The `EnableItem` procedure enables a specified menu item so that it no longer appears
dim and so that the user can choose the menu item.

Note that, if you enable a menu, the `EnableItem` procedure enables the menu title but
only enables those menu items that are not currently disabled as a result of your
application previously calling `DisableItem` and specifying each item's item number.
For example, if all items in your application's Edit menu are enabled, you can disable the
Cut and Copy commands individually using `DisableItem`. If you choose to disable the
entire menu by passing 0 as the `item` parameter to `DisableItem`, the menu and all its
items are disabled. If you then enable the entire menu by passing 0 as the `item`
parameter to `EnableItem`, the menu and its items are enabled, except for the Cut and
Copy commands, which remain disabled. In this case, to enable the Cut and Copy
commands you must enable each one individually using `EnableItem`.

If your application enables a menu using `EnableItem`, it should call `DrawMenuBar` to
update the menu bar's appearance.

SEE ALSO

See "Enabling and Disabling Menu Items" on page 3-58 for examples of enabling items
in a menu.

DisableItem

Use the `DisableItem` procedure to disable a menu item or an entire menu.

```
PROCEDURE DisableItem (theMenu: MenuHandle; item: Integer);
```

theMenu   A handle to the menu record of the menu containing the menu item
               to disable.

item   The item number of the menu item to disable, or 0 to disable the entire
               menu. You cannot individually disable a menu item with an item number
               greater than 31.

               If you specify 0 in the item parameter, the DisableItem procedure
               disables the menu title and all items in the menu, including menu items
               with item numbers greater than 31.

DESCRIPTION

The DisableItem procedure disables a specified menu item so that it appears dim and
cannot be chosen by the user.

If your application disables a menu using DisableItem, your application should call
DrawMenuBar to update the menu bar's appearance.

SEE ALSO

See "Enabling and Disabling Menu Items" on page 3-58 for examples of disabling items
in a menu.

GetMenuItemText

Use the GetMenuItemText procedure to get the text of a specific menu item. The
GetMenuItemText procedure is also available as the GetItem procedure.

```
PROCEDURE GetMenuItemText (theMenu: MenuHandle; item: Integer;
                               VAR itemString: Str255);
```

theMenu   A handle to the menu record of the menu containing the menu item
               whose text you wish to get.

item   The item number of the menu item. The GetMenuItemText procedure
               returns the text of this item.

itemString   The GetMenuItemText procedure returns the text of the menu item in
               this parameter.

DESCRIPTION

The GetMenuItemText procedure returns the text of the specified menu item in the
itemString parameter. Use other Menu Manager routines to get information about
the other characteristics of a menu item.

SetMenuItemText

Use the `SetMenuItemText` procedure to set the text of a specific menu item to a given string. The `SetMenuItemText` procedure is also available as the `SetItem` procedure.

```
PROCEDURE SetMenuItemText (theMenu: MenuHandle; item: Integer;
                            itemString: Str255);
```

theMenu   A handle to the menu record of the menu containing the menu item whose text you wish you to set.

item   The item number of the menu item. The `SetMenuItemText` procedure sets the text of this item.

itemString   The `SetMenuItemText` procedure sets the text of the menu item according to the string specified in the `itemString` parameter. The `SetMenuItemText` procedure does not recognize metacharacters or set any other characteristics of the menu item. The `itemString` parameter can be blank, but it should not be an empty string.

DESCRIPTION

The `SetMenuItemText` procedure sets the text of the specified menu item to the text specified in the `itemString` parameter. The `SetMenuItemText` procedure does not recognize any metacharacters used by the `AppendMenu` and `InsertMenuItem` procedures. Use other Menu Manager routines to set other characteristics of a menu item.

If you set the text of a menu item using the `SetMenuItemText` procedure, you should store the text in a string resource so that your application can be more easily localized.

SEE ALSO

See Listing 3-9 on page 3-59 for an example of setting the text of a menu item.

GetItemStyle

Use the `GetItemStyle` procedure to get the style of the text in a specific menu item.

```
PROCEDURE GetItemStyle (theMenu: MenuHandle; item: Integer;
                        VAR chStyle: Style);
```

theMenu   A handle to the menu record of the menu containing the menu item whose style you wish to get.

item   The item number of the menu item. The `GetItemStyle` procedure returns the style of the text for this item.

chStyle   The GetItemStyle procedure returns the style of the text for this item in
                 the chStyle parameter. The chStyle parameter is a set defined by the
                 Style data type.

```
TYPE
StyleItem = (bold, italic, underline, outline,
             shadow, condense, extend);
Style     = SET OF StyleItem;
```

DESCRIPTION

The GetItemStyle procedure returns the style of the text of the specified menu item in
the chStyle parameter. The returned style can be one or more of the styles defined by
the Style data type, or it is the empty set if the style of the text is Plain.

SetItemStyle

Use the SetItemStyle procedure to set the style of the text in a specific menu item.

```
PROCEDURE SetItemStyle (theMenu: MenuHandle; item: Integer;
                        chStyle: Style);
```

theMenu   A handle to the menu record of the menu containing the menu item
                 whose style you wish to set.

item   The item number of the menu item. The SetItemStyle procedure sets
                 the style of the text for this item.

chStyle   The SetItemStyle procedure sets the style of the text for this item
                 according to the style described by the chStyle parameter. The
                 chStyle parameter is a set defined by the Style data type.

```
TYPE
StyleItem = (bold, italic, underline, outline,
             shadow, condense, extend);
Style     = SET OF StyleItem;
```

                 You can set the style to one or more of the styles defined by the Style
                 data type, or you can set it to Plain by specifying an empty set in the
                 chStyle parameter.

DESCRIPTION

The SetItemStyle procedure sets the style of the text of the specified menu item to the
style or styles defined by the chStyle parameter.

SEE ALSO

See Listing 3-10 on page 3-60 for examples of setting the style of a menu item.

GetItemMark

Use the `GetItemMark` procedure to get the mark of a specific menu item or the menu ID of the submenu associated with the menu item.

```
PROCEDURE GetItemMark (theMenu: MenuHandle; item: Integer;
                         VAR markChar: Char);
```

theMenu   A handle to the menu record of the menu containing the menu item whose mark or submenu you wish to get.

item   The item number of the menu item. The `GetItemMark` procedure returns the mark of this item or, if this item has a submenu associated with it, returns the menu ID of the submenu in the `markChar` parameter.

markChar   The `GetItemMark` procedure returns the mark or the submenu of this item in the `markChar` parameter. A menu item can have a mark or a submenu attached to it, but not both. If this menu item has a marking character, the `GetItemMark` procedure returns the mark. If this menu item has a submenu associated with it, the `GetItemMark` procedure returns the menu ID of the submenu. If the item doesn't have a mark or a submenu, `GetItemMark` returns 0 in this parameter.

DESCRIPTION

If the item has a mark or submenu, the `GetItemMark` procedure returns the mark or the menu ID of the submenu of the specified menu item in the `markChar` parameter (or 0 if the item doesn't have a mark or a submenu).

SetItemMark

Use the `SetItemMark` procedure to set the mark of a specific menu item or to change or set the submenu associated with a menu item.

```
PROCEDURE SetItemMark (theMenu: MenuHandle; item: Integer;
                         markChar: Char);
```

theMenu   A handle to the menu record of the menu containing the menu item whose mark or submenu you wish to set.

item   The item number of the menu item. The `SetItemMark` procedure sets the mark or the submenu of this item.

markChar   The `SetItemMark` procedure sets the mark or submenu of this item according to the information in the `markChar` parameter.

To set the mark of a menu item, specify the marking character in the `markChar` parameter. You can also use one of these constants to specify that the item has no mark, has a checkmark as the marking character, or has the diamond symbol as the marking character:

```
CONST
noMark        = 0;     {no marking character}
checkMark     = $12;   {checkmark}
diamondMark   = $13;   {diamond symbol}
```

To set the submenu associated with this menu item, specify the menu ID of the submenu in the `markChar` parameter.

DESCRIPTION

The `SetItemMark` procedure sets the mark or the submenu of the specified menu item.

SEE ALSO

See Listing 3-11 on page 3-61 for examples of setting the mark of a menu item.

CheckItem

Use the `CheckItem` procedure to set the mark of a specific menu item to a checkmark or to remove a mark from a menu item.

```
PROCEDURE CheckItem (theMenu: MenuHandle; item: Integer;
                     checked: Boolean);
```

theMenu   A handle to the menu record of the menu containing the menu item whose mark you wish to set to a checkmark or whose mark you wish to remove.

item   The item number of the menu item.

checked   The `CheckItem` procedure sets or removes the mark of the item according to the information in the `checked` parameter.

            To set the mark of a menu item to a checkmark, specify TRUE in the `checked` parameter. To remove a checkmark or any other mark from a menu item, specify FALSE in the `checked` parameter.

DESCRIPTION

The `CheckItem` procedure sets the mark of the specified menu item to a checkmark or removes any mark from the menu item.

SEE ALSO

See Listing 3-11 on page 3-61 for examples of setting the mark of a menu item.

GetItemIcon

Use the `GetItemIcon` procedure to get the icon or script code of a specific menu item. If the menu item's keyboard equivalent field contains $1C, the returned number represents the script code of the menu item. Otherwise, the returned number represents the item's icon number.

```
PROCEDURE GetItemIcon (theMenu: MenuHandle; item: Integer;
                           VAR iconIndex: Byte);
```

theMenu   A handle to the menu record of the menu containing the menu item whose icon or script code you wish to get.

item   The item number of the menu item. The `GetItemIcon` procedure returns the icon number or script code of this item.

iconIndex   For menu items that do not specify $1C in the keyboard equivalent field, the `GetItemIcon` procedure returns the icon number of the item's icon in this parameter. The icon number returned in this parameter is a value from 1 through 255 if the menu item has an icon associated with it and is 0 otherwise. You can add 256 to the icon number to generate the resource ID of the `'cicn'`, `'ICON'`, or `'SICN'` resource that describes the icon of the menu item. For example, if the `GetItemIcon` procedure returns 5 in this parameter, then the icon of the menu item is described by an icon resource with resource ID 261.

For menu items that contain $1C in the keyboard equivalent field, the `GetItemIcon` procedure returns the script code of the menu item. The Menu Manager displays the menu item using this script code if the corresponding script system is installed.

DESCRIPTION

The `GetItemIcon` procedure returns the icon number or script code of the specified menu item in the `iconIndex` parameter (or 0 if the item doesn't have an icon or a script code).

SetItemIcon

Use the `SetItemIcon` procedure to set the icon number or script code of a specific menu item. Usually you display menu items in the current system script; however, if needed, you can use the `SetItemIcon` procedure to set the script code of a menu item. For an item's script code to be set, the keyboard equivalent field of the item must contain $1C. If the keyboard equivalent field contains any other value, the `SetItemIcon` procedure interprets the specified number as the item's icon number.

```
PROCEDURE SetItemIcon (theMenu: MenuHandle; item: Integer;
                           iconIndex: Byte);
```

theMenu   A handle to the menu record of the menu containing the menu item
             whose icon (or script code) you wish to set.

item   The item number of the menu item. The SetItemIcon procedure sets
             the icon (or script code) of this item.

iconIndex   If the menu item's keyboard equivalent field does not contain $1C, the
             SetItemIcon procedure sets the icon number of the item's icon to the
             number defined in this parameter. The icon number you specify should
             be a value from 1 through 255 (or from 1 through 254 if the item has a
             small or reduced icon) or 0 if the item does not have an icon.

             The Menu Manager adds 256 to the icon number to generate the resource
             ID of the 'cicn' or 'ICON' resource that describes the icon of the menu
             item. For example, if you specify 5 as the value of the iconIndex
             parameter, when the Menu Manager needs to draw the item, it looks for
             an icon resource with resource ID 261.

             If the menu item's keyboard equivalent field contains $1C, the
             SetItemIcon procedure sets the script code of the menu item to the
             number defined in the iconIndex parameter. The Menu Manager
             displays the menu item using the specified script code if the
             corresponding script system is installed.

             You can specify 0 in the iconIndex parameter to indicate that the item
             uses the current system script and does not have an icon number.

DESCRIPTION

The SetItemIcon procedure sets the icon number or script code of the specified menu
item to the value in the iconIndex parameter.

SEE ALSO

See "Changing the Icon or Script Code of Menu Items" beginning on page 3-62 for
examples of setting the icon of a menu item.

GetItemCmd

Use the GetItemCmd procedure to get the value of the keyboard equivalent field of a
menu item.

```
PROCEDURE GetItemCmd (theMenu: MenuHandle; item: Integer;
                      VAR cmdChar: Char);
```

theMenu   A handle to the menu record of the menu containing the menu item
             whose keyboard equivalent field you wish to get.

item   The item number of the menu item. The GetItemCmd procedure returns
             the keyboard equivalent field of this item.

cmdChar   The value of the item's keyboard equivalent field. The Menu Manager
               uses this value to map keyboard equivalents to menu commands or to
               indicate special characteristics of the menu item.

               If the cmdChar parameter contains $1B, the menu item has a submenu; a
               value of $1C indicates that the item has a script code; a value of $1D
               indicates that the Menu Manager reduces the item's 'ICON' resource;
               and a value of $1E indicates that the item has an 'SICN' resource.

DESCRIPTION

The GetItemCmd procedure returns the value in the keyboard equivalent field of the
specified menu item in the cmdChar parameter (or 0 if the item doesn't have a keyboard
equivalent, submenu, script code, reduced icon, or small icon).

SetItemCmd

Use the SetItemCmd procedure to set the value of the keyboard equivalent field of a
menu item. You usually define the keyboard equivalents and other characteristics of
your menu items in 'MENU' resources rather than using the SetItemCmd procedure.

```
PROCEDURE SetItemCmd (theMenu: MenuHandle; item: Integer;
                      cmdChar: Char);
```

theMenu   A handle to the menu record of the menu containing the menu item
               whose keyboard equivalent field you wish to set.
item   The item number of the menu item. The SetItemCmd procedure sets the
               keyboard equivalent field of this item to the value specified in the
               cmdChar parameter.
cmdChar   The value of the item's keyboard equivalent field. The Menu Manager
               uses this value to map keyboard equivalents to menu commands or to
               define special characteristics of the menu item.

               To indicate that the menu item has a submenu, specify $1B in the
               cmdChar parameter; specify a value of $1C to indicate that the item has a
               script code; specify a value of $1D to indicate that the Menu Manager
               should reduce the item's 'ICON' resource to the size of a small icon; and
               specify a value of $1E to indicate that the item has an 'SICN' resource.

               The values $01 through $1A, as well as $1F and $20, are reserved for use
               by Apple. You should not use any of these reserved values in the
               cmdChar parameter.

DESCRIPTION

The SetItemCmd procedure sets the value in the keyboard equivalent field of the
specified menu item in the cmdChar parameter (you can specify 0 if the item doesn't
have a keyboard equivalent, submenu, script code, reduced icon, or small icon). If you

specify that the item has a submenu, you should provide the menu ID of the submenu as the item's marking character. If you specify that the item has a script code, provide the script code in the icon field of the menu item. If you specify that the item has an 'SICN' or a reduced 'ICON' resource, provide the icon number in the icon field of the item.

Disposing of Menus

If you no longer need a menu in the menu list, you can delete the menu using `DeleteMenu`. You should then release the memory associated with that menu using the `DisposeMenu` procedure if you created the menu using `NewMenu`; otherwise, use the Resource Manager procedure `ReleaseResource`. See the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox* for information on the `ReleaseResource` routine.

DisposeMenu

To release the memory occupied by a menu's associated data structures, use either the `DisposeMenu` procedure or the Resource Manager procedure `ReleaseResource`. Use `DisposeMenu` if you created the menu using `NewMenu`; use `ReleaseResource` if you created the menu using `GetMenu` or read the resource in using `GetNewMBar`.

You should delete the menu from the current menu list using `DeleteMenu` or `ClearMenuBar` before calling the `DisposeMenu` procedure.

```
PROCEDURE DisposeMenu (theMenu: MenuHandle);
```

theMenu   A handle to the menu record of the menu you wish to dispose of.

DESCRIPTION

The `DisposeMenu` procedure releases the memory occupied by the specified menu's menu record. The handle that you pass in the parameter `theMenu` is not valid after `DisposeMenu` returns.

SEE ALSO

To delete a menu from the current menu list, see the description of the `DeleteMenu` procedure on page 3-109.

Counting the Items in a Menu

If your application needs to count the number of items in a menu—for example, in a menu that can contain a variable number of menu items such as the Font menu or Help menu—use the `CountMItems` function.

CountMItems

You can count the number of items in a menu using the CountMItems function.

```
FUNCTION CountMItems (theMenu: MenuHandle): Integer;
```

theMenu   A handle to the menu record of the menu whose items your application
           needs to count.

DESCRIPTION

The CountMItems function counts the number of items in the specified menu and
returns as its function result the number of items in the menu.

Highlighting the Menu Bar

You can highlight (invert) a menu title or the entire menu bar using the FlashMenuBar
procedure. (The HiliteMenu procedure highlights only menu titles.) In most cases
your application should not highlight the menu bar; use HiliteMenu to highlight a
menu title.

The user sets the number of times an enabled menu item flashes using the General
Controls panel. The SetMenuFlash procedure can be used to control the number of
times that menu items blink when the user chooses an enabled menu item; usually you
should not change the setting chosen by the user.

FlashMenuBar

Use the FlashMenuBar procedure to highlight (invert) a menu title or the entire menu
bar. You can call FlashMenuBar twice in a row to make the menu bar blink.

```
PROCEDURE FlashMenuBar (menuID: Integer);
```

menuID   The menu ID of the menu whose title you want to invert. Use 0 in this
           parameter to invert the entire menu bar. If the specified menu ID does not
           exist in the current menu list, the FlashMenuBar procedure inverts the
           entire menu bar.

DESCRIPTION

The FlashMenuBar procedure inverts the title of the specified menu or inverts the
menu bar. To prevent unexpected colors from appearing in the menu bar, you should
not call FlashMenuBar to invert a menu title while the entire menu bar is inverted.

Only one menu title can be inverted at a time. If no menus are currently highlighted, calling FlashMenuBar with a specific menu ID inverts the title of that menu. If you call FlashMenuBar again specifying another menu ID that is different from that of the previously inverted menu title, FlashMenuBar restores the previously highlighted menu to normal and then inverts the title of the specified menu.

SEE ALSO

You can also highlight a menu using the HiliteMenu procedure, described on page 3-119.

SetMenuFlash

Use the SetMenuFlash procedure to set the number of times a menu item blinks when the user chooses an enabled menu item. The user sets this value using the General Controls panel, and in most cases your application should not change the value set by the user.

```
PROCEDURE SetMenuFlash (count: Integer);
```

count   The number of times an enabled menu item should blink when the user chooses it. This value is initially set to 3 by the General Controls panel. A count of 0 disables the blinking. Values greater than 3 can be slow and distracting to the user.

DESCRIPTION

The SetMenuFlash procedure sets the number of times that the Menu Manager causes a menu item to blink when the user chooses an enabled menu item.

The appearance of blinking in a menu item is determined by the menu's menu definition procedure.

ASSEMBLY-LANGUAGE INFORMATION

The global variable MenuFlash contains the current count (number of times) a menu item blinks when chosen by the user.

Recalculating Menu Dimensions

The Menu Manager uses the CalcMenuSize procedure to recalculate the dimensions of a menu whenever its contents have changed. In most cases your application does not need to use the CalcMenuSize procedure.

CalcMenuSize

The `CalcMenuSize` procedure recalculates the horizontal and vertical dimensions of a menu and stores the new values in the `menuWidth` and `menuHeight` fields of the menu record.

```
PROCEDURE CalcMenuSize (theMenu: MenuHandle);
```

theMenu   A handle to the menu record of the menu whose dimensions need recalculating.

DESCRIPTION

The `CalcMenuSize` procedure uses the menu definition procedure of the specified menu to calculate the dimensions of the menu.

Managing Entries in the Menu Color Information Table

The Menu Manager maintains color information about an application's menus in a menu color information table. The standard menu definition procedure defines the standard color for the menu bar, titles of menus, text and characteristics of a menu item, and background color of a displayed menu. You can change any of these colors by adding entries to your application's menu color information table. However, note that in most cases your application should use the default colors for its menus.

You can provide an `'mctb'` resource with resource ID 0 as one of your application's resources if you want to use colors other than the default colors for your application's menu bar and menus. (Or you can provide an `'mctb'` resource with the same resource ID as a `'MENU'` resource to define the color entries for a single menu.) You can also add entries to or delete entries from your application's menu color information table using the `SetMCEntries` and `DeleteMCEntries` procedures. You can get information about an entry using the `GetMCEntry` function. To get or set your application's menu color information table, use the `GetMCInfo` function or `SetMCInfo` procedure. To dispose of your application's menu color information table, use the `DisposeMCInfo` procedure.

Note that the menu color information table uses a format that is different from the standard color table format. "The Menu Color Information Table Record" beginning on page 3-98 describes the format of the menu color information table in detail.

GetMCInfo

Use the `GetMCInfo` function to get a handle to a copy of your application's menu color information table.

```
FUNCTION GetMCInfo: MCTableHandle;
```

DESCRIPTION

The `GetMCInfo` function creates a copy of your application's menu color information table and returns a handle to the copy. If the copy fails, `GetMCInfo` returns `NIL`.

SEE ALSO

See "The Menu Color Information Table Record" beginning on page 3-98 for a description of the format of the menu color information table.

SetMCInfo

Use the `SetMCInfo` procedure to set your application's menu color information table.

```
PROCEDURE SetMCInfo (menuCTbl: MCTableHandle);
```

menuCTbl   A handle to a menu color information table.

DESCRIPTION

The `SetMCInfo` procedure copies the table specified by the `menuCTbl` parameter to your application's menu color information table. If successful, the `SetMCInfo` procedure is responsible for disposing of your application's current menu color information table, so your application does not need to explicitly dispose of the current table.

Your application should call the Memory Manager function `MemError` to determine whether the `SetMCInfo` procedure successfully copied the table. If the `SetMCInfo` procedure cannot successfully copy the table, it does not dispose of the current menu color information table and the `MemError` function returns a nonzero result code. If the `SetMCInfo` procedure is able to successfully copy the table, it disposes of the current menu color information table and the `MemError` function returns the `noErr` result code.

If the menu color information table specifies a new menu bar color or new menu title colors, your application should call `DrawMenuBar` after calling `SetMCInfo`.

Note that `GetNewMBar` does not save your application's current menu color information table. If your application changes menu bars, you can save and restore your application's current menu color information table by calling `GetMCInfo` before `GetNewMBar` and calling `SetMCInfo` afterward.

SEE ALSO

See "The Menu Color Information Table Record" beginning on page 3-98 for a description of the format of the menu color information table. For an example of using the `GetMCInfo` and `SetMCInfo` routines to save and restore menu color information, see Listing 3-6 on page 3-52. See *Inside Macintosh: Memory* for information on the `MemError` function

DisposeMCInfo

Use the `DisposeMCInfo` procedure to dispose of a menu color information table. The `DisposeMCInfo` procedure is also available as the `DispMCInfo` procedure.

```
PROCEDURE DisposeMCInfo (menuCTbl: MCTableHandle);
```

menuCTbl   A handle to a menu color information table.

DESCRIPTION

The `DisposeMCInfo` procedure disposes of the menu color information table referred to by the `menuCTbl` parameter.

GetMCEntry

Use the `GetMCEntry` function to return information about an entry in your application's menu color information table. You can get information about the menu bar entry, a menu title entry, or a menu item entry.

```
FUNCTION GetMCEntry (menuID: Integer; menuItem: Integer)
                     : MCEntryPtr;
```

menuID   The menu ID that the `GetMCEntry` function should use to return
            information about the menu color information table. Specify 0 in the
            `menuID` parameter (and the `menuItem` parameter) to get the menu bar
            entry. Specify the menu ID of a menu in the current menu list in the
            `menuID` parameter and 0 in the `menuItem` parameter to get a specific
            menu title entry. Specify the menu ID of a menu in the current menu list
            in the `menuID` parameter and an item number in the `menuItem`
            parameter to get a specific menu item entry.

menuItem   The menu item that the `GetMCEntry` function should use to return
            information about the menu color information table. If you specify 0 in
            this parameter, `GetMCEntry` returns either the menu bar entry or the
            menu title entry, depending on the value of the `menuID` parameter. If you
            specify the item number of a menu item in this parameter and the menu
            ID of a menu in the current menu list in the `menuID` parameter,
            `GetMCEntry` returns a specific menu item entry.

DESCRIPTION

The `GetMCEntry` function returns a menu bar entry, a menu title entry, or a menu item entry according to the values specified in the `menuID` and `menuItem` parameters. If the `GetMCEntry` function finds the specified entry in your application's menu color information table, it returns a pointer to a record of data type `MCEntry`. If the specified entry is not found, `GetMCEntry` returns `NIL`.

▲   **WARNING**
The menu color information table is relocatable, so the pointer returned by the `GetMCEntry` function may not be valid across routines that may move or purge memory. Your application should make a copy of the menu color entry record if necessary. ▲

SEE ALSO

"The Menu Color Information Table Record" beginning on page 3-98 describes the entries in a menu color information table.

SetMCEntries

Use the `SetMCEntries` procedure to set entries in your application's menu color information table. You can set any or all of your application's menu item entries and menu title entries or the menu bar entry.

```
PROCEDURE SetMCEntries (numEntries: Integer;
                        menuCEntries: MCTablePtr);
```

`numEntries`   The number of entries contained in the array of menu color entry records.

`menuCEntries`
            A pointer to an array of menu color entry records. Specify the number of records in the array in the `numEntries` parameter.

DESCRIPTION

The `SetMCEntries` procedure sets any specified menu bar entry, menu title entry, or menu item entry according to the values specified in the menu color entry records. If an entry already exists for a specified menu color entry, the `SetMCEntries` procedure updates the entry in your application's menu color information table with the new values. If the entry doesn't exist, it is added to your application's menu color information table.

If any of the added entries specify a new menu bar color or new menu title colors, your application should call `DrawMenuBar` to update the menu bar with the new colors.

SPECIAL CONSIDERATIONS

The `SetMCEntries` procedure may move or purge memory. Your application should make sure that the array specified by the `menuCEntries` parameter is nonrelocatable before calling `SetMCEntries`.

"The Menu Color Information Table Record" beginning on page 3-98 describes the
entries in a menu color information table.

DeleteMCEntries

Use the DeleteMCEntries procedure to delete one or all entries for a specific menu
from your application's menu color information table. You can delete a menu item entry,
a menu title entry, the menu bar entry, or all menu item entries of a specific menu. The
DeleteMCEntries procedure is also available as the DelMCEntries procedure.

```
PROCEDURE DeleteMCEntries (menuID: Integer; menuItem: Integer);
```

menuID   The menu ID that the DeleteMCEntries procedure should use to
              determine which entry to delete from the menu color information table.
              Specify 0 in the menuID parameter (and the menuItem parameter) to
              delete the menu bar entry. Specify the menu ID of a menu in the current
              menu list in the menuID parameter and 0 in the menuItem parameter to
              delete a specific menu title entry. Specify the menu ID of a menu in the
              current menu list in the menuID parameter and an item number in the
              menuItem parameter to delete a specific menu item entry.

menuItem   The menu item that the DeleteMCEntries procedure should use to
              determine which entry to delete from the menu color information table. If
              you specify 0 in this parameter, DeleteMCEntries deletes either the
              menu bar entry or menu title entry, depending on the value of the
              menuID parameter. If you specify the item number of a menu item in this
              parameter and the menu ID of a menu in the current menu list in the
              menuID parameter, DeleteMCEntries deletes a specific menu item
              entry. You can also delete all menu item entries for a specific menu from
              your application's menu color information table using this constant:

```
CONST
mctAllItems      = -98;  {delete all menu item entries }
                         { for the specified menu}
```

The DeleteMCEntries procedure deletes a menu bar entry, a menu title entry, a menu
item entry, or all menu item entries of a given menu, according to the values specified in
the menuID and menuItem parameters. If the GetMCEntry function does not find the
specified entry in your application's menu color information table, it does not delete the
entry. Your application should not delete the last entry in your application's menu color
information table.

If any of the deleted entries changes the menu bar color or a menu title color, your
application should call DrawMenuBar to update the menu bar.

Application-Defined Routine

Apple provides a standard menu definition procedure and standard menu bar definition function. The Menu Manager uses the menu definition procedure and menu bar definition function to display and perform basic operations on menus and the menu bar. Although the Menu Manager allows you to provide your own menu bar definition function, Apple recommends that you use the standard menu bar definition function. Similarly, in most cases the standard menu definition procedure should meet the needs of most applications. However, if your application has special needs, you can choose to provide your own menu definition procedure. If you do so, define your menu definition procedure so that it emulates the standard behavior of menus as much as possible. If you define your own menus, they should follow the guidelines described in this chapter and in *Macintosh Human Interface Guidelines.*

The Menu Definition Procedure

The Menu Manager uses the menu definition procedure of a menu to draw the menu items in the menu, to determine which item the user chose from the menu, and to calculate the menu's dimensions. If you provide your own menu definition procedure, it should also perform these tasks.

Apple provides a standard menu definition procedure, stored as a resource in the System file. The standard menu definition procedure is the `'MDEF'` resource with resource ID 0. When you define your menus, you specify the menu definition procedure the Menu Manager should use when managing them. You'll usually want to use the standard menu definition procedure for your application. However, if you need a feature not provided by the standard menu definition procedure (for example, if you want to include more graphics in your menus), you can choose to write your own menu definition procedure.

MyMenuDef

You can provide your own menu definition procedure if you need special features in a menu other than those provided by the standard menu definition procedure. This section describes how to define your own menu definition procedure, defines the parameters passed to your procedure by the Menu Manager, and describes the general actions your procedure should perform.

```
PROCEDURE MyMenuDef (message: Integer; theMenu: MenuHandle;
                     VAR menuRect: Rect; hitPt: Point;
                     VAR whichItem: Integer);
```

message   A number that identifies the operation that the menu definition proce-
            dure should perform. The `message` parameter can contain any one of
            these values:

```
CONST
      mDrawMsg          = 0;  {draw the menu}
      mChooseMsg        = 1;  {tell which item was chosen }
                              { and highlight it}
      mSizeMsg          = 2;  {calculate menu dimensions}
      mPopUpMsg         = 3;  {calculate rectangle of }
                              { the pop-up box}
```

Your menu definition procedure should not respond to any value other than the four constants listed above.

theMenu   A handle to the menu record of the menu that the operation should affect.

menuRect   The rectangle (in global coordinates) in which the menu is located; the Menu Manager provides this information to the menu definition procedure only when the value in the `message` parameter is the `mDrawMsg` or `mChooseMsg` constant.

When the value in the `message` parameter is the `mPopUpMsg` constant, the menu definition procedure should calculate and then return the dimensions of the pop-up box in this parameter. When the value in the `message` parameter is the `mSizeMsg` constant, the menu definition procedure should calculate the horizontal and vertical dimensions of the menu rectangle and store these values in the `menuWidth` and `menuHeight` fields of the menu record.

hitPt   A mouse location (in global coordinates). The Menu Manager provides information in this parameter to the menu definition procedure when the value in the `message` parameter is the `mChooseMsg` or `mPopUpMsg` constant. When the menu definition procedure receives the `mChooseMsg` constant in the `message` parameter, it should determine whether the mouse location specified in the `hitPt` parameter is in an enabled menu item and highlight or unhighlight the item specified in the `whichItem` parameter appropriately. When the menu definition procedure receives the `mPopUpMsg` constant in the `message` parameter, the `hitPt` parameter contains the top-left coordinates of the closed pop-up box, which your procedure can use to calculate the rectangle of the open pop-up box.

whichItem The item number of the last item chosen from this menu (or 0 if an item hasn't been chosen). The Menu Manager provides information in this parameter to the menu definition procedure when the value in the `message` parameter is the `mChooseMsg` constant. When the menu definition procedure receives the `mChooseMsg` constant in the `message` parameter, it should determine whether the mouse location specified in the `hitPt` parameter is in an enabled menu item. If so, the menu definition procedure should unhighlight the item specified by the `whichItem` parameter, highlight the new item, and return the new item number in `whichItem`. If the mouse location isn't in an enabled menu item, the menu definition procedure should unhighlight the item specified by the `whichItem` parameter and return 0 in the `whichItem` parameter.

DESCRIPTION

The Menu Manager calls your menu definition procedure whenever it needs your definition procedure to perform a certain action on a specific menu. The action your menu definition procedure should perform depends on the value of the `message` parameter.

If you provide your own menu definition procedure, store it in a resource of type `'MDEF'` and include its resource ID in the description of each menu that uses your own definition procedure. If you create a menu using `GetMenu` (or `GetNewMBar`), the Menu Manager reads the menu definition procedure into memory and stores a handle to it in the `menuProc` field of the menu's menu record.

If you create a menu using `NewMenu`, the Menu Manager stores a handle to the standard menu definition procedure in the `menuProc` field of the menu's menu record. In this case you must replace the value in the `menuProc` field with a handle to your own procedure and then call the `CalcMenuSize` procedure. If your menu definition procedure is in a resource file, you can get its handle by using the Resource Manager to read it from the resource file into memory. However, note that you should usually store your menus in resources (rather than using `NewMenu`) to make your application easier to localize. See the "Resource Manager" chapter in *Inside Macintosh: More Macintosh Toolbox* for information on the Resource Manager.

The menu definition procedure is responsible for drawing the contents of the menu and its menu items, determining whether the cursor is in a displayed menu, highlighting and unhighlighting menu items, and calculating a menu's dimensions.

When the Menu Manager requests your menu definition procedure to perform an action on a menu, it provides your procedure with a handle to its menu record. This allows your procedure to access the data in the menu record and to use any data in the variable data portion of the menu record to appropriately handle the menu items.

When the Menu Manager creates a menu as a result of an application calling `GetMenu` or `GetNewMBar`, it fills out the `menuID`, `menuProc`, `enableFlags`, `menuTitle`, and `itemDefinitions` fields of the menu record according to its resource definition. If the menu is managed by your menu definition procedure, the Menu Manager calls your procedure (specifying `mSizeMsg`) to calculate and fill in the `menuHeight` and `menuWidth` fields of the menu record. The menu items are described by a variable length field (`itemDefinitions`) in the menu record. Your menu definition procedure can define and use this variable-length data in any manner it chooses.

For pop-up menus that are not implemented as controls, the Menu Manager uses the menu definition procedure to support pop-up menus. If your menu definition procedure supports pop-up menus, it should respond appropriately to the `mPopUpMsg` constant.

The Menu Manager specifies the `mPopUpMsg` constant in the `message` parameter and calls your menu definition procedure whenever it needs to calculate the rectangle bounded by the pop-up box for a pop-up menu that is managed by your menu definition procedure. The parameter `theMenu` contains a handle to the menu record of the pop-up menu, the `hitPt` parameter contains the top-left coordinates of the pop-up box, and `whichItem` contains the previously chosen item. Your menu definition procedure should calculate the rectangle in which the pop-up menu is to appear

and return this rectangle in the `menuRect` parameter. If the menu is so large that it scrolls, return the actual top of the menu in the `whichItem` parameter. For pop-up menus, your menu definition procedure also must place the pop-up menu's scrolling information in the global variables `TopMenuItem` and `AtMenuBottom`. Place in `TopMenuItem` the pixel value of the top of the scrollable menu, and place in `AtMenuBottom` the pixel value of the bottom of the scrollable menu.

Note
Your menu definition procedure should not assume that the A5 register is properly set up, so your procedure can't refer to any of the QuickDraw global variables. ◆

SEE ALSO

For additional information on how your menu definition procedure should respond when it receives the `mDrawMsg`, `mChooseMsg`, or `mSizeMsg` constant in the `message` parameter, see "Writing Your Own Menu Definition Procedure" beginning on page 3-87.

Resources

This section describes the menu (`'MENU'`) resource, menu bar (`'MBAR'`) resource, and menu color information table (`'mctb'`) resource. Usually you should define your menus using `'MENU'` resources, define the menus in your menu bar in an `'MBAR'` resource, and use the `GetNewMBar` function to read in the descriptions of your menus and menu bar.

If you want to use colors other than the default colors in a menu, you can provide an `'mctb'` resource with the same resource ID as its corresponding `'MENU'` resource, or you can provide an `'mctb'` resource with resource ID 0 to define colors for all your menus and your menu bar.

If you choose to provide your own menu definition procedure, you should store your routine in an `'MDEF'` resource.

To create a `'MENU'`, an `'MBAR'`, or an `'mctb'` resource, either you can specify the resource description in an input file and compile the resource using a resoure compiler, such as Rez, or you can directly create your resources in a resource file using a tool such as ResEdit. This section describes the structures of these resources after they are compiled by the Rez resource compiler. If you are interested in creating the Rez input files for these resources, see "Using the Menu Manager," beginning on page 3-41, for detailed information.
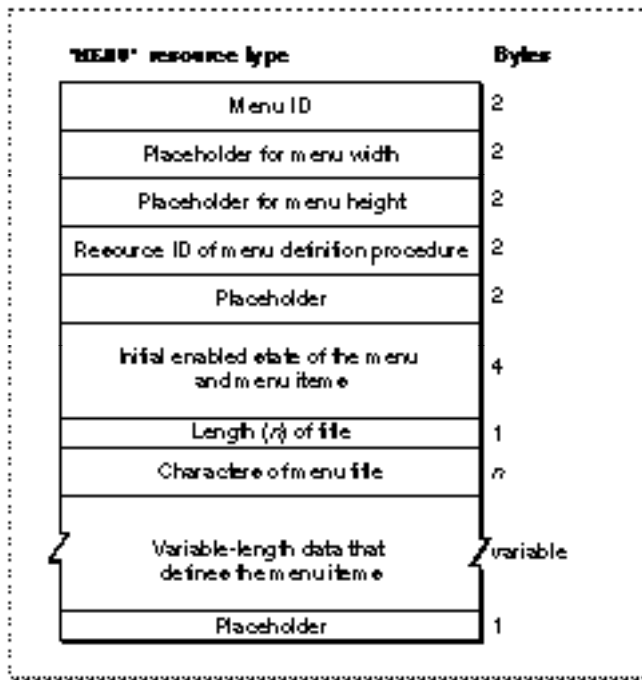
The Menu Resource

You can provide descriptions of your menus in `'MENU'` resources and use the `GetMenu` function or `GetNewMBar` function (if you also provide an `'MBAR'` resource) to read in the descriptions of your menus. After reading in the resource description, the Menu Manager stores the information about specific menus in menu records.

▲ **WARNING**

Menus in a resource must not be purgeable. ▲

Figure 3-37 shows the format of a compiled `'MENU'` resource. See Listing 3-1 on page 3-43 for a description of a `'MENU'` resource in Rez input format.

Figure 3-37   Structure of a compiled menu (`'MENU'`) resource

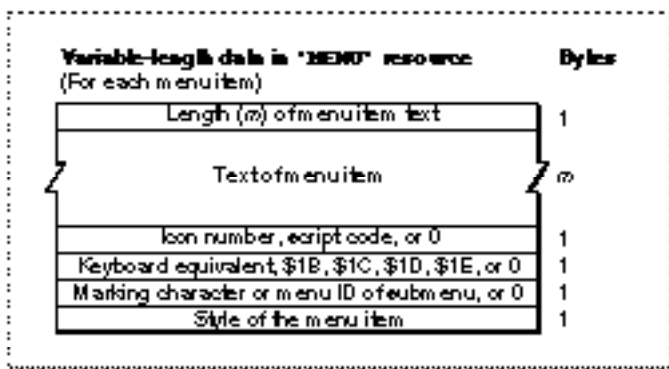A compiled version of a `'MENU'` resource contains the following elements:

■ Menu ID. Each menu in your application should have a unique menu ID. Note that the menu ID does not have to match the resource ID, although by convention most applications assign the same number for a menu's resource ID and menu ID. A negative menu ID indicates a menu belonging to a desk accessory (except for submenus of a desk accessory). A menu ID from 1 through 235 indicates a menu (or submenu) of an application; a menu ID from 236 through 255 indicates a submenu of a desk accessory. Apple reserves the menu ID of 0.

■ Placeholder (two integers containing 0) for the menu's width and height. After reading in the resource data, the Menu Manager requests the menu's menu definition procedure to calculate the width and height of the menu and to store these values in the `menuWidth` and `menuHeight` fields of the menu record.

■ Resource ID of the menu's menu definition procedure. If the integer 0 appears here (as specified by the `textMenuProc` constant in the Rez input file), the Menu Manager uses the standard menu definition procedure to manage the menu. If you provide your own menu definition procedure, its resource ID should appear in these bytes.

After reading in the menu's resource data, the Menu Manager reads in the menu definition procedure, if necessary. The Menu Manager stores a handle to the menu's menu definition procedure in the `menuProc` field of the menu record.

■ Placeholder (an integer containing 0).

■ The initial enabled state of the menu and first 31 menu items. This is a 32-bit value, where bits 1–31 indicate if the corresponding menu item is disabled or enabled, and bit 0 indicates whether the menu is enabled or disabled. The Menu Manager automatically enables menu items greater than 31 when a menu is created.

■ The length (in bytes) of the menu title.

■ The title of the menu.

■ Variable-length data that describes the menu items. If you provide your own menu definition procedure, you can define and provide this variable-length data according to the needs of your procedure. The Menu Manager simply reads in the data for each menu item and stores it as variable data at the end of the menu record. The menu definition procedure is responsible for interpreting the contents of the data. For example, the standard menu definition procedure interprets this data according to the description given in the following paragraphs.

■ Placeholder (a byte containing 0) to indicate the end of the menu item definitions.

If you use the standard menu definition procedure, your 'MENU' resource should describe the menu items in this manner. For each menu item, you need to provide its text, the icon number, the keyboard equivalent or other value ($1B to indicate the menu item has a submenu, $1C to indicate a script code other than the system script for the item's text, $1D to indicate the item's icon should be reduced, or $1E to indicate that an 'SICN' icon should be used), the marking character of the menu item or menu ID of the menu item's submenu, and the font style of the menu item's text. If an item doesn't have a particular characteristic, specify 0 for that characteristic. Figure 3-38 shows the variable-length data portion of a compiled 'MENU' resource that uses the standard menu definition procedure.

Figure 3-38   The variable-length data that describes menu items as defined by the standard menu definition procedure

The variable-length data portion of a compiled version of a 'MENU' resource that uses the standard menu definition procedure contains the following elements:

■ Length (in bytes) of the menu item's text.

■ Text of the menu item.

■ Icon number, script code, or 0 (as specified by the noicon constant in a Rez input file) if the menu item doesn't contain an icon and uses the system script. The icon number is a number from 1 through 255 (or from 1 through 254 for small or reduced icons). The Menu Manager adds 256 to the icon number to generate the resource ID of the menu item's icon. If a menu item has an icon, you should also provide a 'cicn' or an 'ICON' resource with the resource ID equal to the icon number plus 256. If you want the Menu Manager to reduce an 'ICON' resource to the size of a small icon, also provide the value $1D in the keyboard equivalent field. If you provide an 'SICN' resource, provide $1E in the keyboard equivalent field. Otherwise, the Menu Manager looks first for a 'cicn' resource with the calculated resource ID and uses that icon. If you want the Menu Manager to draw the item's text in a script other than the system script, specify the script code here and also provide $1C in the keyboard equivalent field. If the script system for the specified script is installed, the Menu Manager draws the item's text using that script. An item that is drawn in a script other than the system script cannot also have an icon.

■ Keyboard equivalent (specified as a 1-byte character), the value $1B (as specified by the constant hierarchicalMenu in a Rez input file) if the item has a submenu, the value $1C if the item uses a script other than the system script, or 0 (as specified by the nokey constant in a Rez input file) if the item has neither a keyboard equivalent nor a submenu and uses the system script. A menu item can have a keyboard equivalent, a submenu, a small icon, a reduced icon, or a script code, but not more than one of these characteristics. For items containing icons, you can provide $1D in this field if you want the Menu Manager to reduce an 'ICON' resource to the size of a small icon. Provide $1E if you want the Menu Manager to use an 'SICN' resource for the item's icon. The values $01 through $1A as well as $1F and $20 are reserved for use by Apple; your application should not use any of these reserved values in this field.

■ Marking character, the menu ID of the item's submenu, or 0 (as specified by the nomark constant in a Rez input file) if the item has neither a mark nor a submenu. A menu item can have a mark or a submenu, but not both. Submenus of an application should have menu IDs from 1 through 235; submenus of a desk accessory should have menu IDs from 236 through 255.

■ Font style of the menu item. The constants bold, italic, plain, outline, and shadow can be used in a Rez input file to define their corresponding styles.

If you provide your own menu definition procedure, you should use the same format for your resource descriptions of menus as shown in Figure 3-37. You can use the same format or a format of your choosing to describe menu items. You can also use bits 1–31 of the enableFlags field of the menu record as you choose; however, bit 0 must still indicate whether the menu is enabled or disabled.

The Menu Bar Resource

You can describe the order and number of menus in your menu bar in an 'MBAR' resource, and you can describe your menus in 'MENU' resources. If you do so, you can use the GetNewMBar function to read in the descriptions of your menus and create a new menu list. The Menu Manager stores information about your application's menu bar in a menu list. Figure 3-39 shows the format of a compiled 'MBAR' resource. (See Listing 3-4 on page 3-49 for a description of an 'MBAR' resource in Rez input format.)

Figure 3-39   Structure of a compiled menu bar ('MBAR') resource
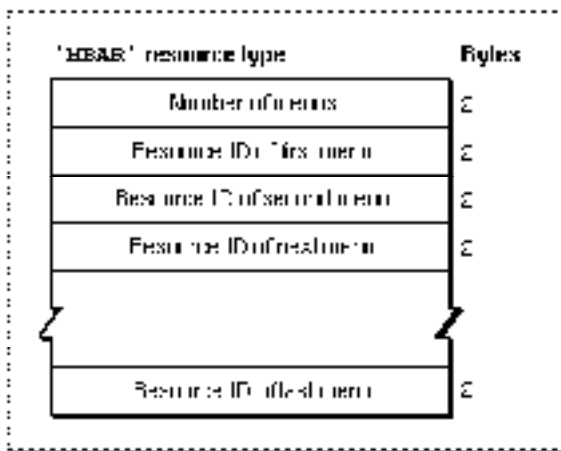
A compiled version of an 'MBAR' resource contains the following elements:

■ Number of menus described by this menu bar.

■ A variable number (the amount should match the number declared in the first 2 bytes) of resource IDs; each resource ID should identify a 'MENU' resource.

If you use the GetNewMBar function, the Menu Manager places the menus in the menu bar according to the order that they appear in the 'MBAR' resource.

## The Menu Color Information Table Resource

To use colors other than the default colors in a menu, provide a **menu color information table** ('mctb') **resource** with the same resource ID as its corresponding 'MENU' resource. You can also choose to provide an 'mctb' resource with resource ID 0 to define colors for all your menus and your menu bar. Note that you should usually use the default colors provided by the Menu Manager.

The Menu Manager stores color information about your application's menus and menu bar in a menu color information table. If you provide an 'mctb' resource with resource ID 0, the Menu Manager reads the resource in when your application calls InitMenus and stores the information in your application's menu color information table. If you provide an 'mctb' resource with the same resource ID as a 'MENU' resource, when you

use `GetMenu` to read in the resource description of the menu (or `GetNewMBar` to read in all menus in the menu bar), the Menu Manager also reads in any associated `'mctb'` resource (if it exists). "The Menu Color Information Table Record" beginning on page 3-98 describes the format of the menu color information table.

Figure 3-40 shows the format of a compiled `'mctb'` resource.

Figure 3-40   Structure of a compiled menu color information table (`'mctb'`) resource
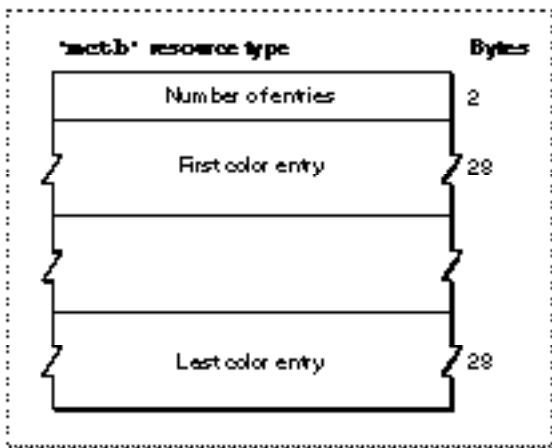

A compiled version of an `'mctb'` resource contains the following elements:

■ a count of the number of menu color entry descriptions

■ a variable number of menu color entries

A color entry defines colors for various parts of the menu and menu bar. Figure 3-41 on the next page shows the format of a compiled menu color entry in an `'mctb'` resource.

Each menu color entry in an `'mctb'` resource contains the following:

■ A menu ID to indicate that this entry is either a menu item entry or menu title entry, 0 to indicate that this entry is a menu bar entry, or –99 to indicate that this is the last entry in this resource.

■ An item number to indicate that this entry is a menu item entry, or 0 to indicate that this is either a menu title or menu bar entry. Together, the menu ID and menu item determine how the type of menu color entry is described. See Table 3-7 on page 3-100 for a complete description of how the menu ID and menu item specifications define the type of menu color entry.

■ RGB1: for a menu bar entry, the default color for menu titles; for a menu title entry, the title color of a specific menu; for a menu item entry, the mark color for a specific item.

■ RGB2: for a menu bar entry, the default background color of a displayed menu; for a menu title entry, the default color for the menu bar; for a menu item entry, the color for the text of a specific item.

Figure 3-41   Structure of a menu color entry in an `'mctb'` resource
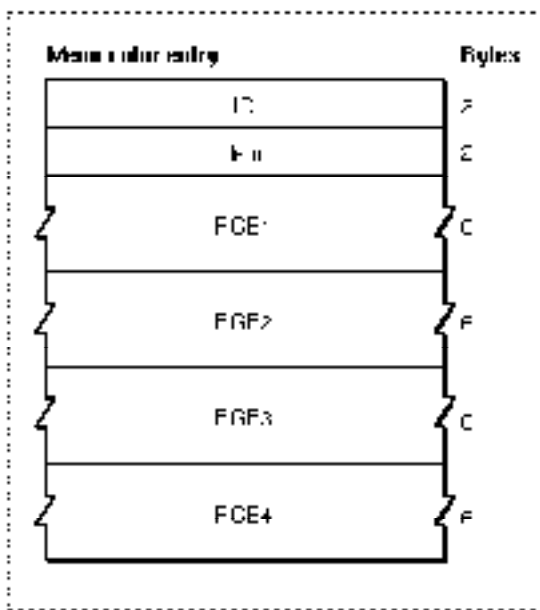
■ RGB3: for a menu bar entry, the default color of items in a displayed menu; for a menu title entry, the default color for items in a specific menu; for a menu item entry, the color for the keyboard equivalent of a specific item.

■ RGB4: for a menu bar entry, the default color of the menu bar; for a menu title entry, the background color of a specific menu; for a menu item entry, the background color of a specific menu.

The Menu Definition Procedure Resource

If you provide your own menu definition procedure, you should store it in a resource of type `'MDEF'`. Provide as the resource data the compiled or assembled code of your menu definition procedure. The entry point of your procedure must be at the beginning of the resource data.

If you define your menus in `'MENU'` resources (and use the GetMenu or GetNewMBar function), you specify the menu definition procedure that the Menu Manager should use to manage the menu in the `'MENU'` resource. If you use the NewMenu function (instead of `'MENU'` resources), your application must explicitly replace the handle to the standard menu definition procedure in the menuProc field of the menu record with a handle to the desired menu definition procedure.

Summary of the Menu Manager

Pascal Summary

Constants

```
CONST
     noMark          = 0;  {menu item doesn't have a marking character}

     {values for the message parameter to the menu definition procedure}
     mDrawMsg        = 0;  {draw the menu items of a menu}
     mChooseMsg      = 1;  {highlight or unhighlight a menu item as }
                           { appropriate if the cursor is in a menu item}
     mSizeMsg        = 2;  {calculate the dimensions of a menu}
     mPopUpMsg       = 3;  {calculate the open pop-up box rectangle}

     textMenuProc    = 0;  {resource ID of standard menu definition }
                           { procedure}
     hMenuCmd        = 27; {constant ($1B) specified as keyboard equivalent }
                           { to indicate a menu item has a submenu}
     hierMenu        = -1; {constant used with InsertMenu routine to insert }
                           { a submenu or pop-up menu into the submenu }
                           { portion of the current menu list}
     mctAllItems     = -98;{search for all items with the given ID}
     mctLastIDIndic  = -99;{last menu color table entry has this value }
                           { in the ID field of the entry}
```

Data Types

```
TYPE
     MenuInfo =                 {menu record}
     RECORD
        menuID:    Integer;     {number that identifies the menu}
        menuWidth:  Integer;    {width (in pixels) of the menu}
        menuHeight: Integer;    {height (in pixels) of the menu}
        menuProc:   Handle;     {menu definition procedure}
        enableFlags:LongInt;    {indicates whether menu and }
                                { menu items are enabled}
```

```
    menuData:   Str255;     {title of menu}
    {itemDefinitions}       {variable-length data that }
                            { defines the menu items}
END;

MenuPtr     = ^MenuInfo;   {pointer to a menu record}
MenuHandle  = ^MenuPtr;    {handle to a menu record}

MCEntry =                  {menu color entry record}
RECORD
   mctID:       Integer;   {menu ID or 0 for menu bar}
   mctItem:     Integer;   {menu item number or 0 for }
                           { menu title}
   mctRGB1:     RGBColor;  {usage depends on mctID and }
                           { mctItem}
   mctRGB2:     RGBColor;  {usage depends on mctID and }
                           { mctItem}
   mctRGB3:     RGBColor;  {usage depends on mctID and }
                           { mctItem}
   mctRGB4:     RGBColor;  {usage depends on mctID and }
                           { mctItem}
   mctReserved:Integer;    {reserved}
END;

MCEntryPtr  = ^MCEntry;    {pointer to a menu color entry record}

MCTable     = ARRAY[0..0] OF MCEntry;  {menu color table}
MCTablePtr  = ^MCTable;        {pointer to a menu color table}
MCTableHandle = ^MCTablePtr;  {handle to a menu color table}
```

Menu Manager Routines

Initializing the Menu Manager

```
PROCEDURE InitMenus;
PROCEDURE InitProcMenu       (resID: Integer);
```

Creating Menus

```
FUNCTION NewMenu             (menuID: Integer; menuTitle: Str255)
                              : MenuHandle;
FUNCTION GetMenu             (resourceID: Integer): MenuHandle;
```

Adding Menus to and Removing Menus From the Current Menu List

```
PROCEDURE InsertMenu          (theMenu: MenuHandle; beforeID: Integer);
PROCEDURE DeleteMenu          (menuID: Integer);
PROCEDURE ClearMenuBar;
```

Getting a Menu Bar Description From an 'MBAR' Resource

```
FUNCTION GetNewMBar           (menuBarID: Integer): Handle;
```

Getting and Setting the Menu Bar

```
FUNCTION GetMenuBar: Handle;
PROCEDURE SetMenuBar          (menuList: Handle);
FUNCTION GetMBarHeight: Integer;
```

Drawing the Menu Bar

```
PROCEDURE DrawMenuBar;
PROCEDURE InvalMenuBar;
```

Responding to the User's Choice of a Menu Command

```
FUNCTION MenuSelect           (startPt: Point): LongInt;
FUNCTION MenuKey              (ch: Char): LongInt;
FUNCTION MenuChoice: LongInt;
PROCEDURE HiliteMenu          (menuID: Integer);
FUNCTION PopUpMenuSelect      (menu: MenuHandle;
                               Top: Integer; Left: Integer;
                               PopUpItem: Integer): LongInt;
PROCEDURE SystemMenu          (menuResult: LongInt);
FUNCTION SystemEdit           (editCmd: Integer): Boolean;
```

Getting a Handle to a Menu Record

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
FUNCTION GetMenuHandle         (menuID: Integer): MenuHandle;
FUNCTION HMGetHelpMenuHandle
                              (VAR mh: MenuHandle): OSErr;
```

Adding and Deleting Menu Items

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
PROCEDURE AppendMenu          (menu: MenuHandle; data: Str255);
PROCEDURE InsertMenuItem      (theMenu: MenuHandle; itemString: Str255;
                               afterItem: Integer);
```

```
PROCEDURE DeleteMenuItem     (theMenu: MenuHandle; item: Integer);
PROCEDURE AppendResMenu      (theMenu: MenuHandle; theType: ResType);
PROCEDURE InsertResMenu      (theMenu: MenuHandle; theType: ResType;
                              afterItem: Integer);
```

Getting and Setting the Appearance of Menu Items

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
PROCEDURE EnableItem         (theMenu: MenuHandle; item: Integer);
PROCEDURE DisableItem        (theMenu: MenuHandle; item: Integer);
PROCEDURE GetMenuItemText    (theMenu: MenuHandle; item: Integer;
                              VAR itemString: Str255);
PROCEDURE SetMenuItemText    (theMenu: MenuHandle; item: Integer;
                              itemString: Str255);
PROCEDURE GetItemStyle       (theMenu: MenuHandle; item: Integer;
                              VAR chStyle: Style);
PROCEDURE SetItemStyle       (theMenu: MenuHandle; item: Integer;
                              chStyle: Style);
PROCEDURE GetItemMark        (theMenu: MenuHandle; item: Integer;
                              VAR markChar: Char);
PROCEDURE SetItemMark        (theMenu: MenuHandle; item: Integer;
                              markChar: Char);
PROCEDURE CheckItem          (theMenu: MenuHandle; item: Integer;
                              checked: Boolean);
PROCEDURE GetItemIcon        (theMenu: MenuHandle; item: Integer;
                              VAR iconIndex: Byte);
PROCEDURE SetItemIcon        (theMenu: MenuHandle; item: Integer;
                              iconIndex: Byte);
PROCEDURE GetItemCmd         (theMenu: MenuHandle; item: Integer;
                              VAR cmdChar: CHAR);
PROCEDURE SetItemCmd         (theMenu: MenuHandle; item: Integer;
                              cmdChar: CHAR);
```

Disposing of Menus

```
PROCEDURE DisposeMenu        (theMenu: MenuHandle);
```

Counting the Items in a Menu

```
FUNCTION CountMItems         (theMenu: MenuHandle): Integer;
```

Highlighting the Menu Bar

```
PROCEDURE FlashMenuBar       (menuID: Integer);
PROCEDURE SetMenuFlash       (count: Integer);
```

Recalculating Menu Dimensions

```
PROCEDURE CalcMenuSize        (theMenu: MenuHandle);
```

Managing Entries in the Menu Color Information Table

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
FUNCTION GetMCInfo: MCTableHandle;
PROCEDURE SetMCInfo           (menuCTbl: MCTableHandle);
PROCEDURE DisposeMCInfo       (menuCTbl: MCTableHandle);
FUNCTION GetMCEntry           (menuID: Integer; menuItem: Integer)
                               : MCEntryPtr;
PROCEDURE SetMCEntries        (numEntries: Integer;
                               menuCEntries: MCTablePtr);
PROCEDURE DeleteMCEntries     (menuID: Integer; menuItem: Integer);
```

Application-Defined Routine

```
PROCEDURE MyMenuDef           (message: Integer; theMenu: MenuHandle;
                               VAR menuRect: Rect; hitPt: Point;
                               VAR whichItem: Integer);
```

C Summary

Constants

```
enum {
     #define noMark '\0'  /*menu item doesn't have a marking character*/

     /*values for the message parameter to the menu definition procedure*/
     mDrawMsg      = 0,  /*draw the menu items of a menu*/
     mChooseMsg    = 1,  /*highlight or unhighlight a menu item as */
                         /* appropriate if the cursor is in a menu item*/
     mSizeMsg      = 2,  /*calculate the dimensions of a menu*/
     mPopUpMsg     = 3,  /*calculate the open pop-up box rectangle*/

     textMenuProc  = 0,  /*resource ID of standard menu definition */
                         /* procedure*/
     hMenuCmd      = 27, /*constant ($1B) specified as keyboard */
                         /* equivalent to indicate an item has a submenu*/
     hierMenu      = -1, /*constant used with InsertMenu to insert */
                         /* a submenu or pop-up menu into the submenu */
                         /* portion of the current menu list*/
```

```
      mctAllItems    = -98,/*search for all items with the given ID*/
      mctLastIDIndic = -99 /*last menu color table entry has this value */
                           /* in the ID field of the entry*/
   };
```

Data Types

```
struct MenuInfo {                /*menu record*/
        short       menuID;        /*number that identifies the menu*/
        short       menuWidth;     /*width (in pixels) of the menu*/
        short       menuHeight;    /*height (in pixels) of the menu*/
        Handle      menuProc;      /*menu definition procedure*/
        long        enableFlags;   /*indicates whether menu and */
                                   /* menu items are enabled*/
        Str255      menuData;      /*title of menu*/
        /*itemDefinitions*/        /*variable-length data that */
                                   /* defines the menu items*/
     };

typedef struct MenuInfo MenuInfo;        /*pointer to a menu record*/
typedef MenuInfo *MenuPtr, **MenuHandle;  /*handle to a menu record*/

struct MCEntry {              /*menu color entry record*/
        short       mctID;         /*menu ID or 0 for menu bar*/
        short       mctItem;       /*menu item number or 0 for */
                                   /* menu title*/
        RGBColor    mctRGB1;       /*usage depends on mctID and */
                                   /* mctItem*/
        RGBColor    mctRGB2;       /*usage depends on mctID and */
                                   /* mctItem*/
        RGBColor    mctRGB3;       /*usage depends on mctID and */
                                   /* mctItem*/
        RGBColor    mctRGB4;       /*usage depends on mctID and */
                                   /* mctItem*/
        short       mctReserved;   /*reserved*/
     };

typedef struct MCEntry MCEntry;
typedef MCEntry *MCEntryPtr;            /*pointer to a menu color entry record*/
                                       /*menu color table*/
typedef MCEntry MCTable[1], *MCTablePtr, **MCTableHandle;
```

Menu Manager Routines

Initializing the Menu Manager

```
pascal void InitMenus      (void);
pascal void InitProcMenu   (short resID);
```

Creating Menus

```
pascal MenuHandle NewMenu  (short menuID, const Str255 menuTitle);
pascal MenuHandle GetMenu  (short resourceID);
```

Adding Menus to and Removing Menus From the Current Menu List

```
pascal void InsertMenu     (MenuHandle theMenu, short beforeID);
pascal void DeleteMenu     (short menuID);
pascal void ClearMenuBar   (void);
```

Getting a Menu Bar Description From an 'MBAR' Resource

```
pascal Handle GetNewMBar   (short menuBarID);
```

Getting and Setting the Menu Bar

```
pascal Handle GetMenuBar   (void);
pascal void SetMenuBar     (Handle menuList);
#define GetMBarHeight()    (* (short*) 0x0BAA)
```

Drawing the Menu Bar

```
pascal void DrawMenuBar    (void);
pascal void InvalMenuBar   (void);
```

Responding to the User's Choice of a Menu Command

```
pascal long MenuSelect       (Point startPt);
pascal long MenuKey          (short ch);
pascal long MenuChoice       (void);
pascal void HiliteMenu       (short menuID);
pascal long PopUpMenuSelect (MenuHandle menu, short top, short left,
                              short popUpItem);
pascal void SystemMenu       (long menuResult);
pascal Boolean SystemEdit   (short editCmd);
```

Getting a Handle to a Menu Record

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal MenuHandle GetMenuHandle
                            (short menuID);
pascal OSErr HMGetHelpMenuHandle
                            (MenuHandle *mh);
```

Adding and Deleting Menu Items

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal void AppendMenu       (MenuHandle menu, ConstStr255Param data);
pascal void InsertMenuItem   (MenuHandle theMenu,
                              ConstStr255Param itemString,
                              short afterItem);
pascal void DeleteMenuItem   (MenuHandle theMenu, short item);
pascal void AppendResMenu    (MenuHandle theMenu, ResType theType);
pascal void InsertResMenu    (MenuHandle theMenu, ResType theType,
                              short afterItem);
```

Getting and Setting the Appearance of Menu Items

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal void EnableItem       (MenuHandle theMenu, short item);

pascal void DisableItem      (MenuHandle theMenu, short item);

pascal void GetMenuItemText  (MenuHandle theMenu, short item,
                              Str255 itemString);

pascal void SetMenuItemText  (MenuHandle theMenu, short item,
                              ConstStr255Param itemString);

pascal void GetItemStyle     (MenuHandle theMenu, short item,
                              Style *chStyle);

pascal void SetItemStyle     (MenuHandle theMenu, short item, short chStyle);

pascal void GetItemMark      (MenuHandle theMenu, short item,
                              short *markChar);

pascal void SetItemMark      (MenuHandle theMenu, short item,
                              short markChar);

pascal void CheckItem        (MenuHandle theMenu, short item,
                              Boolean checked);

pascal void GetItemIcon      (MenuHandle theMenu, short item,
                              short *iconIndex);

pascal void SetItemIcon      (MenuHandle theMenu, short item,
                              short iconIndex);
```

3

Menu Manager

```
pascal void GetItemCmd       (MenuHandle theMenu, short item, short
                              *cmdChar);
pascal void SetItemCmd       (MenuHandle theMenu, short item, short cmdChar);
```

Disposing of Menus

```
pascal void DisposeMenu      (MenuHandle theMenu);
```

Counting the Items in a Menu

```
pascal short CountMItems     (MenuHandle theMenu);
```

Highlighting the Menu Bar

```
pascal void FlashMenuBar     (short menuID);
pascal void SetMenuFlash     (short count);
```

Recalculating Menu Dimensions

```
pascal void CalcMenuSize     (MenuHandle theMenu);
```

Managing Entries in the Menu Color Information Table

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal MCTableHandle GetMCInfo(void);
pascal void SetMCInfo        (MCTableHandle menuCTbl);
pascal void DisposeMCInfo    (MCTableHandle menuCTbl);
pascal MCEntryPtr GetMCEntry (short menuID, short menuItem);
pascal void SetMCEntries     (short numEntries, MCTablePtr menuCEntries);
pascal void DeleteMCEntries  (short menuID, short menuItem);
```

Application-Defined Routine

```
pascal void MyMenuDef        (short message, MenuHandle theMenu,
                              Rect *menuRect, Point hitPt,
                              short *whichItem);
```

Assembly-Language Summary

Data Structures

The Menu Information Data Structure

0	menuID	word	number that identifies the menu
2	menuWidth	word	width (in pixels) of the menu
4	menuHeight	word	height (in pixels) of the menu
6	menuDefHandle	long	menu definition procedure
10	menuEnable	long	enable flags
14	menuData	256 bytes	menu title followed by menu item information

Global Variables

AtMenuBottom	The pixel value at the bottom of the scrollable menu.
MBarEnable	Contains 0 if all menus in the current menu bar belong to an application; contains a nonzero value if all menus belong to a desk accessory.
MBarHeight	Contains current height of the menu bar, in pixels.
MBarHook	Address of routine that MenuSelect calls repeatedly while the mouse button is down.
MenuCInfo	Contains a handle to application's menu color information table.
MenuDisable	Contains the menu ID and item number of the last item chosen, regardless of whether the item was disabled or enabled.
MenuFlash	Contains the current count (number of times) a menu item blinks when chosen by the user.
MenuHook	Address of routine that MenuSelect calls after a menu title is highlighted and the menu rectangle is calculated but before the menu is drawn.
TheMenu	Contains the menu ID of the highlighted menu in the menu bar.
TopMenuItem	The pixel value at the top of the scrollable menu.

Result Codes

noErr	0	No error
paramErr	−50	Error in parameter list
memFullErr	−108	Not enough room in heap zone
resNotFound	−192	Unable to read resource
hmHelpManagerNotInited	−855	Help menu not set up

3

Menu Manager