# 3

# MENUS

## Demonstration Programs: Menus1 and Menus2

## Introduction — Types of Menus

A menu is a user interface element that allows the user to view, or choose from, a list of choices and commands provided by your application. There are basically three types of menus:

- ***Pull-Down Menus.*** A pull-down menu comprises a menu title, displayed in the menu bar, and one or more menu items.
- ***Submenus.*** A submenu is a menu that is attached to another menu. A menu to which a submenu is attached is referred to as a **hierarchical menu**.
- ***Pop-Up Menus.*** A pop-up menu is a menu that does not appear in the menu bar but rather appears on another part of the screen..

## Pull-Down Menus

### Menu Definition Functions and Menu Bar Definition Functions

The Menu Manager uses the following to display, and to perform basic operations on, menus and the menu bar:

- ***Menu Definition Function.*** When you define a menu, you must specify the required menu definition function (MDEF). The Menu Manager uses that MDEF to draw the menu items in a menu, determine which item the user chose, etc. An MDEF thus determines the look and behaviour of menus.
- ***Menu Bar Definition Function.*** The Menu Manager uses the menu bar definition function (MBDF) to draw and clear the menu bar, determine whether the cursor is currently within the menu bar or any currently displayed menu, highlight menu titles, etc. A menu bar definition function thus determines the look and behaviour of the menu bar.

### Standard Menu and Menu Bar Definition Functions

The system software provides a standard MDEF and a standard MBDF. The standard MDEF is the `'MDEF'` resource with a resource ID of 63. The standard MBDF is the `'MBDF'` resource with a resource ID of 63.

Ordinarily, your application will specify the standard definition functions; however, as with most other elements of the Macintosh user interface, the option is available to write your own custom definition function if you need to provide features not available in the standard definition functions.

## The Menu Bar and Menus

### The Menu Bar

The menu bar extends across the top of the screen and is high enough to display menu titles in the height of the large system font (Mac OS 8/9) or system font (Mac OS X).

Generally, the menu bar should always be visible. If you want to hide the menu bar for some reason, you should provide a method (for example, a keyboard equivalent) to allow the user to make the menu bar reappear.

#### The 'MBAR' Resource

Each application has its own menu bar, which is defined by an `'MBAR'` resource. This resource lists the order and resource ID of each menu appearing in your menu bar. Your application's `'MBAR'` resource should be defined such that the Mac OS 8/9 Apple menu or Mac OS X Application menu (see below) is the first menu in the menu bar, with the **File** menu being the next. For Mac OS 8/9, the **Help** menu and the Mac OS 8/9 Application menu (see below) do not need to be defined in the `'MBAR'` resource, since the Menu Manager automatically adds them to the menu bar when the application calls `GetNewMBar` provided that your menu bar includes the Apple menu.

## Menus

All Macintosh applications should ordinarily provide, as a minimum, the Mac OS 8/9 Apple menu (for Mac OS 8/9) or Mac OS X Application menu (for Mac OS X), a **File** menu, and a **Window** menu (see Chapter 16). If your application is not document-oriented, the **File** menu may be renamed to something more appropriate.

Your application can disable any menu, which causes the Menu Manager to dim that menu's title and all associated menu items. The menu items can also be disabled individually. Your application should specify whether menu items are enabled or disabled when it first defines and creates a menu and can enable or disable items at any time thereafter.

#### The 'MENU' Resource

For each menu, you define the menu title and the individual characteristics of its menu items in a `'MENU'` resource.

#### The 'xmnu' Resource

For each menu, you may also define an `'xmnu'` (extended menu) resource. The `'xmnu'` resource is, in effect, an extension of the `'MENU'` resource required to provide for additional menu features. . Note that you do not need to provide this resource if you do not require these additional features. An `'xmnu'` resource must have the same ID as the `'MENU'` resource it extends.

## Menu Items

A menu item can contain text or a **divider**. On Mac OS 8/9 the divider is a line extending the full width of the menu. On Mac OS X it is simply an empty space, like a menu item with no text. Each menu item, other than a divider, can have a number of characteristics as follows:

- An icon, small icon, reduced icon, colour icon, or an icon from an icon family[1] to the left of the menu item's text.
- A checkmark or other **marking character** indicating the status of the menu item or the mode it controls.
- The symbols for the item's keyboard equivalent. (An item that has a keyboard equivalent cannot have a submenu, a small icon or a reduced icon.)

[1] The various icon types are described at Chapter 13.

· A triangular indicator to the right of a menu item's text to indicate that the item has a submenu. (An item that has a submenu cannot have a keyboard equivalent, a small icon or a reduced icon.)

· A font style (bold, italic, etc.) for the menu item's text.

· The text of the menu item.

· The ellipsis character (**…**) as the last character in the text of the menu item, indicating that, before executing the command, the application will display a dialog requesting more information from the user. (The ellipsis character should not be used in menu items that display informational dialogs or a confirmational alert.)

· A dimmed appearance when the application disables the item. (When the menu title is dimmed, all menu items in that menu are also dimmed.)

## *Groups of Menu Items*

Where appropriate, menu items should be grouped, with each group separated by a divider. For example, a menu can contain commands that perform actions and commands that set attributes. The action commands that are logically related should be grouped, as should attribute commands that are interdependent. The attribute commands that are mutually exclusive, and those that form accumulating attributes (for example, **Bold**, **Italic** and **Underline**), should also be grouped.

## *Keyboard Equivalents for Menu Commands*

The Menu Manager provides support for **keyboard equivalents**[2]. Your application can detect a keyboard equivalent by examining the `modifiers` field of the event structure, first determining whether the Command key was pressed at the time of the event. If a keyboard equivalent is detected, your application typically calls `MenuEvent`, which maps the keyboard equivalent character contained in the specified event structure to its corresponding menu and menu item and returns the menu ID and the chosen menu item.

### *Reserved Command-Key Equivalents*

Apple reserves the following Command-key equivalents, which should be used in the **File** and **Edit** menus of your application:

| *Keys* | *Command* | *Menu* |
|---|---|---|
| Command-A | **Select All** | **Edit** |
| Command-C | **Copy** | **Edit** |
| Command-N | **New** | **File** |
| Command-H | **Hide <appname>** | Application (Mac OS X) |
| Command-M | **Minimize Window** | **Window** (Mac OS X) |
| Command-O | **Open…** | **File** |
| Command-P | **Print…** | **File** |
| Command-Q | **Quit** | **File** (Mac OS 8/9)<br>Application (Mac OS X) |
| Command-S | **Save** | **File** |
| Command-V | **Paste** | **Edit** |
| Command-W | **Close** | **File** |
| Command-X | **Cut** | **Edit** |
| Command-Z | **Undo** | **Edit** |

[2] A **keyboard equivalent** is any combination of the Command key, optionally one or more modifier keys (Shift, Option, Control), and another key. A **Command-key equivalent** such as Command-C is thus, by definition, also a keyboard equivalent.

Other common keyboard equivalents are:

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

## *The Mac OS 8/9 Apple Menu and Mac OS X Application Menu*

On Mac OS 8/9, the Mac OS 8/9 Apple Menu is the first menu in your application.  On Mac OS X, the Mac OS X Application Menu (see Fig 1) is the first menu.

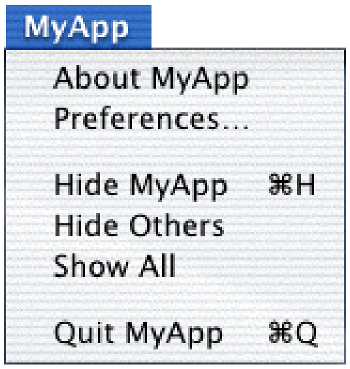


**FIG 1 - MAC OS X APPLICATION MENU**

Typically, applications provide an **About** command as the first menu item in the Apple (Mac OS 8/9) and Mac OS X Application menus.  On Mac OS 8/9, the remaining items are controlled by the contents of the Apple Menu Items folder in the System folder.  On Mac OS X, the remaining items are the default items automatically included in the system-created Mac OS X Application menu.  Mac OS X Application menu items are general to the application, that is, they are items that are not specific to a document or other window.

To create your application's Mac OS 8/9 Apple menu for Mac OS 8/9, you simply define the Apple menu title and the characteristics of your application's **About** command in a `'MENU'` resource.  When your application is run on Mac OS 8/9, the contents of the Apple Menu Items folder are automatically added to the Apple menu.

The Apple menu `'MENU'` resource will also cause the **About** command to be inserted in the Mac OS X Application menu when the application is run on Mac OS X.

When the user chooses the **About** command on Mac OS 8/9, your application should display a dialog or an alert containing your application's name, version number, copyright information, any other information as required.  On Mac OS X, your application should display a modeless dialog containing the application's version and copyright information, as prescribed in Aqua Human Interface Guidelines.

## *The File Menu*

The standard **File** menu contains commands related to document management plus, on Mac OS 8/9, the **Quit** command.  (On Mac OS X, the **Quit** command is located in the Application menu (see Fig 1).)  The standard commands (see Fig 2) should be supported by your application where appropriate.  Any other commands added to the menu should pertain to the management of documents.  The actions your application should take when **File** menu commands are chosen are detailed at Chapter 15 and Chapter 18.

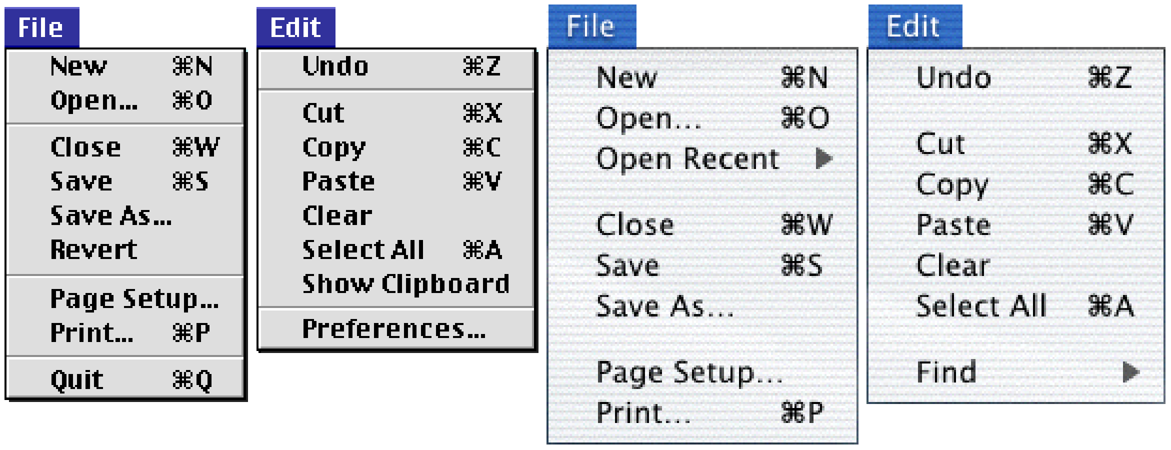


FIG 2 - THE STANDARD FILE AND EDIT MENUS

## The Edit Menu

The standard **Edit** menu (see Fig 2) provides commands related to the editing of a document's contents, to copying data between different applications using the Clipboard and, on Mac OS 8/9, to showing and hiding the Clipboard. For Mac OS 8/9 only, the standard **Edit** menu also standardises the location of the **Preferences...** command, which, when chosen, should invoke a dialog that enables the user to set application-specific preferences. (On Mac OS X, the **Preferences...** command is located in the Mac OS X Application menu.)[3]

All Macintosh applications which support text entry, including text entry in edit text items in dialogs, should include the standard editing commands (**Undo**, **Cut**, **Copy**, **Paste** and **Clear**). An additional word or phrase should be added to **Undo** to clarify exactly what action your application will reverse.

Other commands may be added if they are related to editing or changing the contents of your application's documents.

## The Mac OS 8/9 System-Managed Menus

On Mac OS 8/9, two menus, namely the Mac OS 8/9 Application menu and the **Help** menu, are added automatically by the Menu Manager and are often referred to as the **system-managed menus**.

### The Mac OS 8/9 Application Menu

When the user chooses an item from the Mac OS 8/9 Application menu, the Menu Manager handles the event as appropriate. For example, if the user chooses another application, your application is sent to the background and receives a suspend event.

### The Mac OS 8/9 Help Menu

Applications written for Mac OS 8/9 using the Classic API have the option of programmatically appending an item (or items) to the end of the **Help** menu, and of programmatically detecting the user's choice of that item, so as to give the user access to help texts provided by the application. This option is not available in the Carbon API.

Carbon applications may use **Apple Help**, which was introduced with Mac OS 8.6, to provide application help. Apple Help documentation and tools are included in an Apple Help Software Development Kit (SDK), which is available at <http:/developer.apple.com/sdk>. Amongst other things, the documentation describes how to create an Apple Guide file which, when located in the same folder as your application, will cause the system to install a help menu item (or items) in the **Help** menu. The menu at the left at Fig 3 show the

[3] The implementation of **Preferences** commands is demonstrated at the demonstration program at Chapter 19.

**Help** menu as it normally appears. The menus at the right at Fig 3 show the **Help** menu as it appears when the Apple Guide file is present.

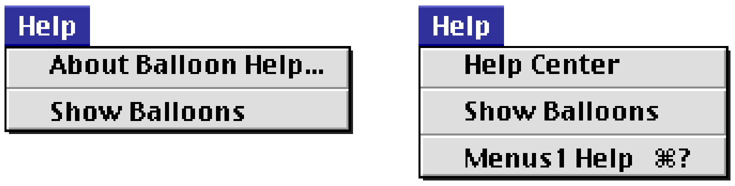


FIG 3 - THE MAC OS 8/9 HELP MENU - EFFECT OF THE APPLE GUIDE FILE

### Mac OS Help Menus

For Mac OS X, your application must itself create the **Help** menu (using the function `HMGetHelpMenu`), insert the required item, or items, in that menu, and respond to the user choosing items in the menu.

### Font Menus

If your application has a **Font** menu, you should list in that menu the names of all currently available fonts (that is, all those residing in the Fonts folder in the System folder). Fonts may be added to the **Font** menu using `AppendResMenu` or `InsertResMenu`. However, a better alternative is to use the relatively new Menu Manager function `CreateStandardFontMenu` to build either a hierarchical or non-hierarchical **Font** menu. (A hierarchical **Font** menu is one in which the styles available in each font family (regular, bold, italic, etc.) appear in a submenu attached to the menu item containing the font family name.)

To indicate which font is currently in use in a non-hierachical **Font** menu, your application should add a checkmark to the left of the font's name in the Font menu. If the current selection contains more than one font, a dash should be placed next to the name of each font the selection contains. When the user starts entering text at the insertion point, your application should display text in the current font.

To indicate which font is currently in use in a hierachical **Font** menu, your application should place a checkmark next to the font in the submenu and a dash next to the menu item to which the submenu is attached.

#### Font Attributes

Separate menus should be used to accommodate lists of font attributes such as styles and sizes.

#### WYSIWYG Font Menus

The function `SetMenuItemFontID` allows you to easily set up a **Font** menu with each item being drawn in the actual font.

## Hierarchical Menus

A hierarchical menu is a menu which has a submenu attached to it. You should use a submenu only when you have more menus than fit in the menu bar. There should only ever be one hierarchical level, that is, there should be only one level of submenus. A menu item that is the title of a submenu should clearly represent the choices the submenu contains.

Hierarchical menus work best for providing a submenu of attributes.

## Pop-Up Menus

Pop-up menus work well when your application needs to present several choices to the user and it is acceptable to hide these choices until the menu is opened. (Other methods of displaying choices are checkboxes and radio buttons.) Pop-up menus should not be used for multiple choice lists or as a way to provide more commands. They should contain attributes rather than actions; accordingly, Command-key equivalents should not be used in pop-up menus.

The standard pop-up menu is actually implemented as a **control**, specifically, **the pop-up menu button control**. Its appearance (see Fig 4) and behaviour is thus determined by a pop-up menu button control definition function.

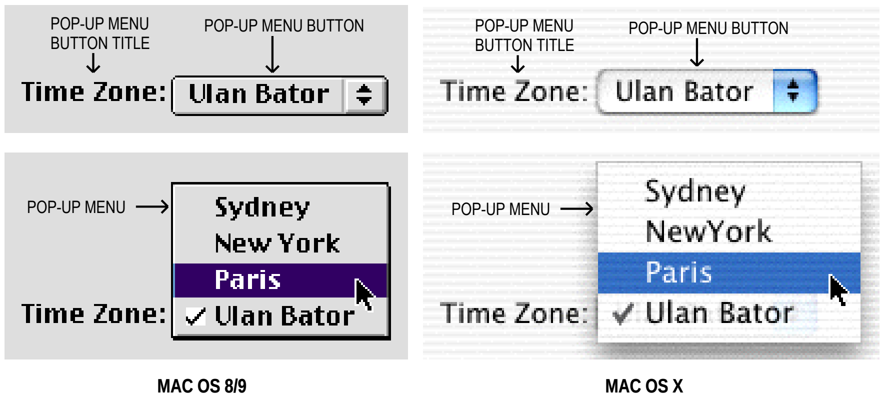


FIG 4 - POP-UP MENU BUTTON (EXAMPLE)

Because pop-up menus are implemented as controls, they are addressed at Chapter 7. Further information in this chapter will be limited to the provision of the `'MENU'` resource required by the pop-up menu button control.

## *Menu Objects, Menu IDs and Item Numbers, Command IDs, and Menu Lists*

### *The Menu Object*

The Menu Manager maintains information about individual menus in opaque data structures known as **menu objects**. The data type `MenuHandle` is defined as a pointer to a menu object:

```
typedef struct OpaqueMenuHandle* MenuHandle;
```

Note that the data type `MenuHandle` is equivalent to the newer data type `MenuRef`:

```
typedef MenuHandle MenuRef;
```

***Carbon Note***

A major change introduced in Carbon is that some commonly used data structures are now **opaque**, meaning that their internal structure is hidden to applications. Directly referencing fields within these structures is no longer possible, and special new **accessor functions** must be used instead.

As an example, the Classic API equivalent of the menu object is the `MenuInfo` structure, which is defined as follows:

```
struct MenuInfo
{
  MenuID  menuID;
  short   menuWidth;
  short   menuHeight;
  Handle  menuProc;
  long    enableFlags;
  Str255  menuData;
};
typedef struct MenuInfo MenuInfo;
typedef MenuInfo *MenuPtr;
typedef MenuPtr *MenuHandle;
```

In the Classic API, your application can determine the menu width by directly accessing the `menuWidth` field like this:

```
MenuHandle menuHdl;
SInt16     width;

menuHdl = GetMenuHandle(mFile);  // Get handle to MenuInfo structure.
width   = (**menuHdl).menuWidth;
```

In Carbon, you must use the accessor function `GetMenuWidth` to obtain the menu width from a menu object:

```
MenuRef menuRef;
SInt16  width;

menuRef = GetMenuRef(mFile);    // Get reference to menu object.
width   = GetMenuWidth(menuRef);
```

The following accessor functions are provided to access the information in menu objects:

| ***Function*** | ***Description*** |
|---|---|
| `GetMenuID` | Gets the menu ID of the specified menu. |
| `SetMenuID` | Sets the menu ID of the specified menu. |
| `GetMenuWidth` | Gets the horizontal dimensions, in pixels, of the specified menu. |
| `SetMenuWidth` | Sets the horizontal dimensions, in pixels, of the specified menu. |
| `GetMenuHeight` | Gets the vertical dimensions, in pixels, of the specified menu. |
| `SetMenuHeight` | Sets the vertical dimensions, in pixels, of the specified menu. |
| `GetMenuTitle` | Gets the title of the specified menu. |
| `SetMenuTitle` | Sets the title of the specified menu. |
| `SetMenuTitleWithCFString` | |
| `GetMenuDefinition` | Gets a pointer to a custom menu definition function that has already been associated with the menu. (There is no way to get a pointer to the system menu definition function.) |
| `SetMenuDefinition` | Sends a dispose message to the current menu definition and an init message to the new definition. |

You typically specify most of the information in a menu object in a `'MENU'` resource. When you create a menu, the Menu Manager creates a menu object for the menu and returns a reference to that object. The

Menu Manager automatically updates the menu object when you make any changes to the menu programmatically.

### Menu IDs and Item Numbers

To refer to a menu, you usually use either the menu's ID or the reference to the menu's menu object. Accordingly, you must assign a **menu ID** to each menu in your application as follows:

- Pull-down menus must use a menu ID greater than 0.
- Submenus of an application may use a menu ID in the range 1 to 32767.

To refer to a menu item, you use the item's **item number**. Item numbers in a menu start at 1.

### Command IDs

The **command ID**, a unique value that you set to identify a menu item, is an alternative way of referring to a specific menu item in an application's menus.

### The Menu List

The **menu list**, a structure private to the Menu Manager, contains references to the menu objects of one or more menus (although a menu list can, in fact, be empty). The end of a menu list contains references to the menu objects of submenus and pop-up menus, if any, the phrase "submenu portion of the menu list" referring to this portion of the list.

At application launch, the Menu Manager creates the menu list. The menu list is initially empty but changes as your application adds menus to it or removes menus from it programmatically.

## Creating Your Application's Menus

### 'MBAR', 'MENU', and 'xmnu' Resources

As stated at Chapter 1, you can provide a textual, formal description of resources in a file and then use a resource compiler such as Rez to compile the description into a resource, or you can create resource descriptions using a resource editor such as Resorcerer. This book assumes the use of Resorcerer.

When creating resources using Resorcerer, it is advisable that you refer to a diagram and description of the structure of the resource and relate that to the various items in the Resorcerer editing windows. Accordingly, the following describes the structure of those resources associated with the creation of menus.

#### Structure of a Compiled 'MBAR' Resource

Fig 5 shows the structure of a compiled `'MBAR'` resource. The number of menu resource IDs should match the number of menus declared in the first two bytes.

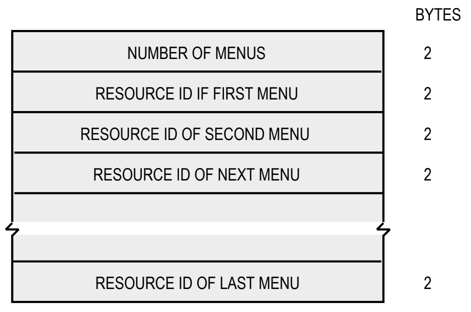
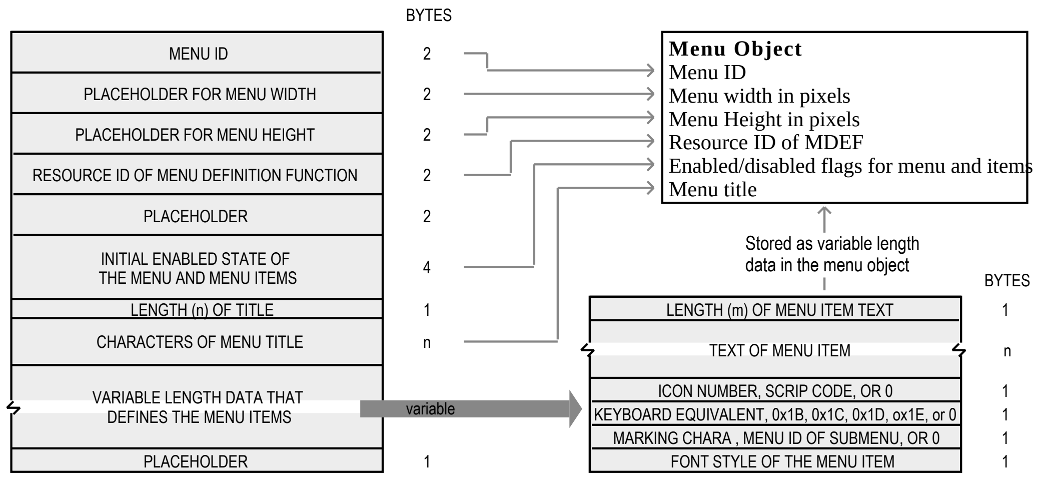


FIG 5 - STRUCTURE OF A COMPILED 'MBAR' RESOURCE

## Structure of a Compiled 'MENU' Resource

Fig 6 shows the structure of a compiled `'MENU'` resource (and its variable length data) and how it "feeds" the menu object.



BYTES
MENU ID 2
PLACEHOLDER FOR MENU WIDTH 2
PLACEHOLDER FOR MENU HEIGHT 2
RESOURCE ID OF MENU DEFINITION FUNCTION 2
PLACEHOLDER 2
INITIAL ENABLED STATE OF THE MENU AND MENU ITEMS 4
LENGTH (n) OF TITLE 1
CHARACTERS OF MENU TITLE n
VARIABLE LENGTH DATA THAT DEFINES THE MENU ITEMS
variable
PLACEHOLDER 1
Menu Object
Menu ID
Menu width in pixels
Menu Height in pixels
Resource ID of MDEF
Enabled/disabled flags for menu and items
Menu title
Stored as variable length data in the menu object
BYTES
LENGTH (m) OF MENU ITEM TEXT 1
TEXT OF MENU ITEM n
ICON NUMBER, SCRIP CODE, OR 0 1
KEYBOARD EQUIVALENT, 0x1B, 0x1C, 0x1D, ox1E, or 0 1
MARKING CHARA , MENU ID OF SUBMENU, OR 0 1
FONT STYLE OF THE MENU ITEM 1

STRUCTURE OF A COMPILED MENU ('MENU') RESOURCE        VARIABLE LENGTH DATA FOR EACH MENU ITEM

FIG 6 - STRUCTURE OF A COMPILED MENU ('MENU') RESOURCE AND ITS VARIABLE LENGTH DATA

The following describes the main fields of the `'MENU'` resource:

| Field | Description |
|---|---|
| MENU ID | The menu's unique identification number. Note that the number assigned to the menu ID and the resource ID do not have to be identical, though it is advisable that these numbers be the same.<br>A menu ID from 1 to 235 indicates a menu (or submenu) of an application. Apple reserves the menu ID of 0. |
| PLACEHOLDER FOR MENU WIDTH<br>PLACEHOLDER FOR MENU HEIGHT | After reading in the resource data, the Menu Manager requests the menu's MDEF to calculate the width and height of the menu and store these values in the menu object. |
| RESOURCE ID OF MENU DEFINITION FUNCTION | If the integer 63 appears here, the standard MDEF will be used. The MDEF is read in after the menu's resource data is read in. The Menu Manager stores a handle to the MDEF in the menu object. |
| INITIAL ENABLED STATE OF THE MENU AND MENU ITEMS | A value whose bits indicate if the corresponding menu item is enabled or disabled, with bit 0 indicating whether the menu as a whole is enabled or disabled. |
| VARIABLE LENGTH DATA THAT DEFINES THE MENU ITEMS | The Menu Manager stores the variable length data for each menu item at the end of the menu object (see Fig 6). |

The following describes the main fields of the variable length data for each menu item. Note that various alternatives apply to the icon number, keyboard equivalent, and marking character fields. For example, a menu item can have a keyboard equivalent or a submenu, but not both.

| Field | Description |
|---|---|
| ICON NUMBER, SCRIPTCODE, OR 0 | ICON NUMBER<br>A number from 1 to 255 (or from 1 to 254 for small or reduced icons).<br>If the menu item specifies an icon, you must provide a `'cicn'` (colour icon) or `'ICON'` resource with a resource ID equal to the icon number plus 256. If you want an `'ICON'` resource to be reduced to the size of a small icon (`'SICN'`), or if you want the size of a `'cicn'` resource reduced by half, assign the value `0x1D` to the keyboard equivalent field (see below). If you want a `'SICN'` resource, assign the value `0x1E`.<br>The Menu Manager looks first for a `'cicn'` resource with the calculated resource ID. In the Carbon era, colour icons are much to be preferred.<br>SCRIPTCODE (Not used when the 'MENU' resource is extended with an 'xmnu' resource)<br>Specify the script code[4] here if you want the item's text to be drawn in a script other than the system script, and also provide `0x1C` in the keyboard equivalent field (see below).<br>*When the `'MENU'` resource is extended by an `'xmnu'` resource, the script code should be set in the text encoding field of the `'xmnu'` resource.*<br>0<br>Specifies that the menu item does not contain an icon and uses the system script. |
| KEYBOARD EQUIVALENT, 0X1B, 0X1C, 0X1D, OX1E, OR 0 | KEYBOARD EQUIVALENT<br>Specified as a one-byte character and, actually, a Command-key equivalent only.<br>The Command-key equivalent can be extended with modifier key (Shift, Option, Control) constants in the modifier keys field of the extended menu (`'xmnu'` ) resource (see below).<br>0x1B<br>Specifies that the menu item has a submenu. (The menu ID of the submenu should be assigned to the marking character field (see below).)<br>0x1C (Not used when the 'MENU' resource is extended with an 'xmnu' resource)<br>Specifies that the item uses a script other than the system script. (The script code should be assigned to the icon number field (see above).)<br>*When the `'MENU'` resource is extended by an `'xmnu'` resource, the script code should be set in the text encoding field of the `'xmnu'` resource.*<br>0x1D<br>For menu items containing icons, causes `'ICON'` resources to be reduced to the size of a small icon, or `'cicn'` resources to be reduced by half.<br>0x1E<br>Specifies that you want the Menu Manager to use a small icon (`'SICN'`) resource for the item's icon. (The small icon's resource ID should be assigned to the icon number field (see above).)<br>0<br>Specifies that the menu item has neither a keyboard equivalent nor a submenu and uses the system script. |
| MARKING CHARACTER, MENU ID OF SUBMENU, OR 0 | MARKING CHARACTER<br>Special characters, such as the checkmark and diamond characters are available to indicate the marks associated with a menu item.<br>MENU ID OF SUBMENU<br>Submenus of an application must have menu IDs from 1 to 235. Submenus of a driver must have menu IDs from 236 to 255.<br>0<br>Specifies that the item has neither a mark nor a submenu. |
| FONT STYLE OF THE MENU ITEM | Specifies whether the font style of the menu item should be plain, or any combination of bold, italic, outline, and shadow. |

[4] A **script system** consists of keyboard resources (which provide for text input in any language from any keyboard) international resources (which contain information specific to a particular language, such as its date and time formats, sorting order, and word-break rules), and fonts (that is, sets of glyphs, which are visual representations of characters). A **script code** is a numeric value indicating a particular Mac OS script system. Constants (e.g., `smRoman`, `smJapanese`) are defined for each of the script codes recognized by the Mac OS. The constant for the script code for the system script system is `smSystemScript`.

## Structure of a Compiled 'xmnu' Resource

The `'xmnu'` resource provides for the additional features, for example, support for extended modifier keys, command IDs, etc. Fig 7 shows the structure of a compiled `'xmnu'` resource and an individual menu item entry in that resource.

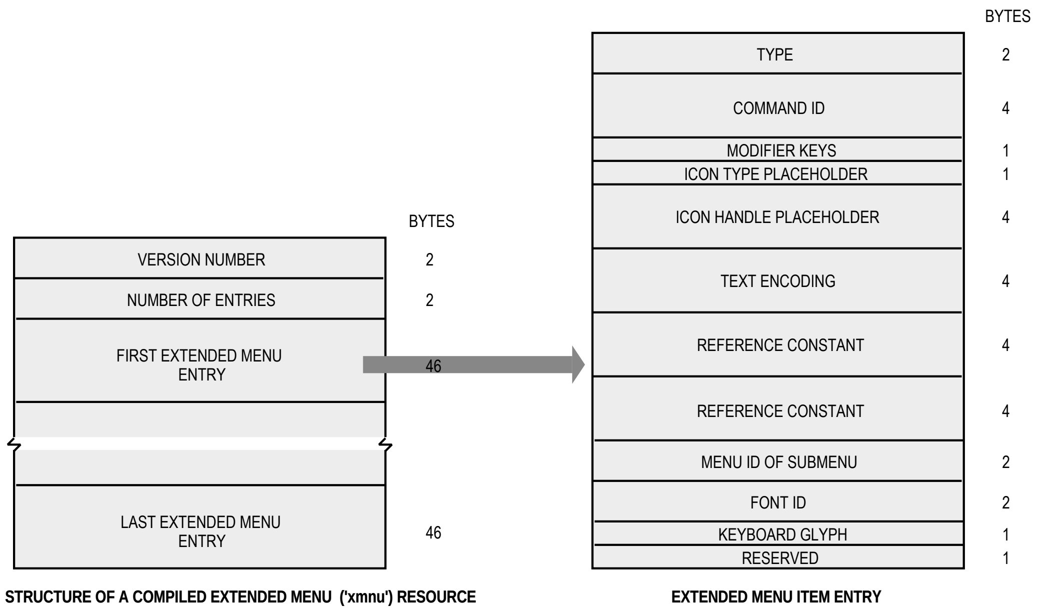


FIG 7 - STRUCTURE OF A COMPILED EXTENDED MENU ('xmnu') RESOURCE AND AN EXTENDED MENU ITEM ENTRY

The following describes the fields of a compiled `'xmnu'` resource:

| ***Field*** | ***Description*** |
|---|---|
| VERSION NUMBER | Version of the resource. |
| NUMBER OF ENTRIES | Number of entries (extended menu item structures) in the resource. |
| FIRST EXTENDED MENU ENTRY<br>…<br>LAST EXTENDED MENU ENTRY | A number of extended menu item structures (see below). |

Each entry in a `'xmnu'` resource corresponds to a menu item. The following describes the main fields of an extended menu item entry.

| ***Field*** | ***Description*** |
|---|---|
| TYPE | Specifies whether there is extended information for the item. 1 indicates that there is extended information for the item, causing the rest of the entry to be read in. 0 indicates that there is no information for the item, causing the Menu Manager to skip the rest of the entry. |
| COMMAND ID | A unique value used to identify the menu item (as opposed to referring to the item using the menu ID and item number).<br>This value may be ascertained via a call to `GetMenuItemCommandID`.<br>A command ID may be assigned to a menu item programmatically via a call to `SetMenuItemCommandID`. |
| MODIFIER KEYS | Specifies the modifier keys used in a keyboard equivalent to select a menu item.<br>The current modifier keys may be ascertained via a call to `GetMenuItemModifiers`.<br>Modifier keys may be assigned to a menu item programmatically via a call to `SetMenuItemModifiers`. |
| ICON TYPE PLACEHOLDER<br>ICON HANDLE PLACEHOLDER | (Reserved. Set to 0.)<br>(Reserved. Set to 0.) |

| | |
|---|---|
| TEXT ENCODING | Specifies the text encoding for the menu item text.<br>This field of the `'xmnu'` resource should be used instead of setting the keyboard equivalent field in the `'MENU'` resource to `0x1C` and the icon number field to the script code.<br>If you want to use the system script, assign -1. If you want to use the current script, assign -2.<br>The current text encoding may be ascertained via a call to `GetMenuItemTextEncoding`.<br>Text encoding may be assigned to a menu item programmatically via a call to `SetMenuItemTextEncoding`. |
| REFERENCE CONSTANT | Any value an application wants to store.<br>The current value may be ascertained via a call to `GetMenuItemRefCon`.<br>Reference constants may be assigned to a menu item programmatically via a call to `SetMenuItemRefCon`. |
| REFERENCE CONSTANT | Any additional value an application wants to store.<br>The getter and setter functions relating to this second reference constant are not available in Carbon. If you wish to associate data with a menu item you should use the functions which are available for that purpose (see Associating Data With Menu Items, below). |
| MENU ID OF SUBMENU | A value between 1 and 235, identifying the submenu.<br>The current submenu ID may be acertained via a call to `GetMenuItemHierarchicalID`.<br>The menu ID of a submenu may be assigned to a menu item programmatically via a call to `SetMenuItemHierarchicalID`. This, in effect, attaches a submenu to the menu item. |
| FONT ID | The ID of the font family. If this value is 0, then the large system font ID (Mac OS 8/9) or system font (Mac OS X) is used.<br>The current font ID may be acertained via a call to `GetMenuItemFontID`.<br>The font ID of a menu item may be set programmatically via a call to `SetMenuItemFontID`. |
| KEYBOARD GLYPH | A symbol representing a menu item's modifier key.<br>The current keyboard glyph may be ascertained via a call to `GetMenuItemGlyph`.<br>If the value in this field is zero, the keyboard glyph uses the keyboard font. (A glyph is a visual representation of a character.) You can override the character code to be displayed with a substitute glyph by assigning a non-zero value to this field.<br>The keyboard glyph of a menu item may be set programmatically via a call to `SetMenuItemKeyGlyph`. |

The information in an `'xmnu'` resource is set for specified menu items; it is not necessary to create an extended menu entry for all menu items in a menu.

It is not necessary to provide `'xmnu'` resources if your application's menus do not require the additional features provided by this resource.

## Creating 'MBAR', 'MENU', and 'xmnu' Resources Using Resorcerer

As previously stated, when creating resources using Resorcerer, it is advisable that you refer to a diagram and description of the structure of the resource and relate that to the various items in the Resorcerer editing windows. The following assumes that approach.

### Creating 'MBAR' Resources

Fig 8 shows an `'MBAR'` resource containing seven menus being created with Resorcerer. The first three entries would be, respectively, the Apple, **File**, and **Edit** menus.

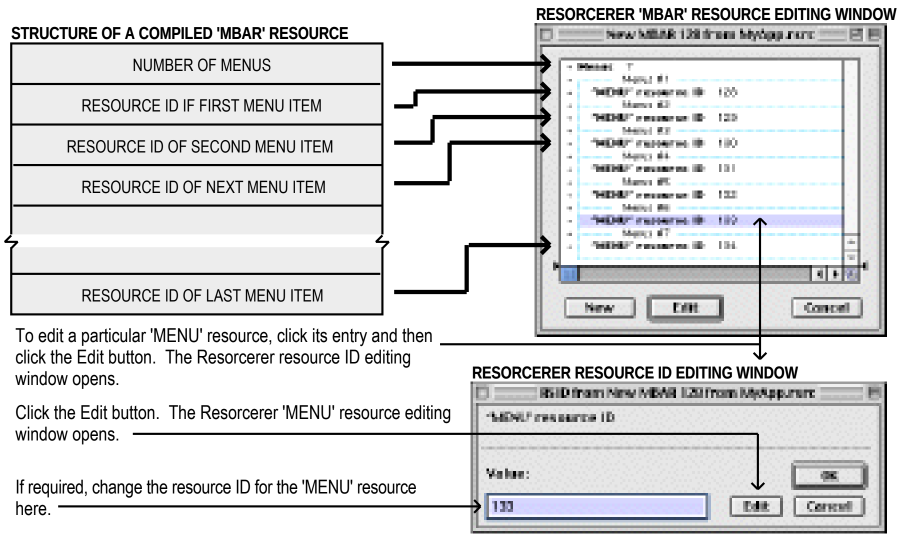

STRUCTURE OF A COMPILED 'MBAR' RESOURCE
NUMBER OF MENUS
RESOURCE ID IF FIRST MENU ITEM
RESOURCE ID OF SECOND MENU ITEM
RESOURCE ID OF NEXT MENU ITEM
RESOURCE ID OF LAST MENU ITEM
RESORCERER 'MBAR' RESOURCE EDITING WINDOW
New
Edit
Cancel
To edit a particular 'MENU' resource, click its entry and then click the Edit button. The Resorcerer resource ID editing window opens.
RESORCERER RESOURCE ID EDITING WINDOW
Click the Edit button. The Resorcerer 'MENU' resource editing window opens.
If required, change the resource ID for the 'MENU' resource here.
Value:
133
OK
Edit
Cancel

**FIG 8 - CREATING AN 'MBAR' RESOURCE USING RESORCERER**

## *Creating 'MENU' Resources*

Fig 9 shows an imaginary **View** menu with the **Full Screen** menu item being edited. This menu item has been assigned a keyboard equivalent (more specifically, a Command-key equivalent); accordingly, the **Key Equiv:** radio button has been clicked and the character **F** has been entered as the Command-key equivalent. The menu item has also been assigned a marking character (a checkmark).

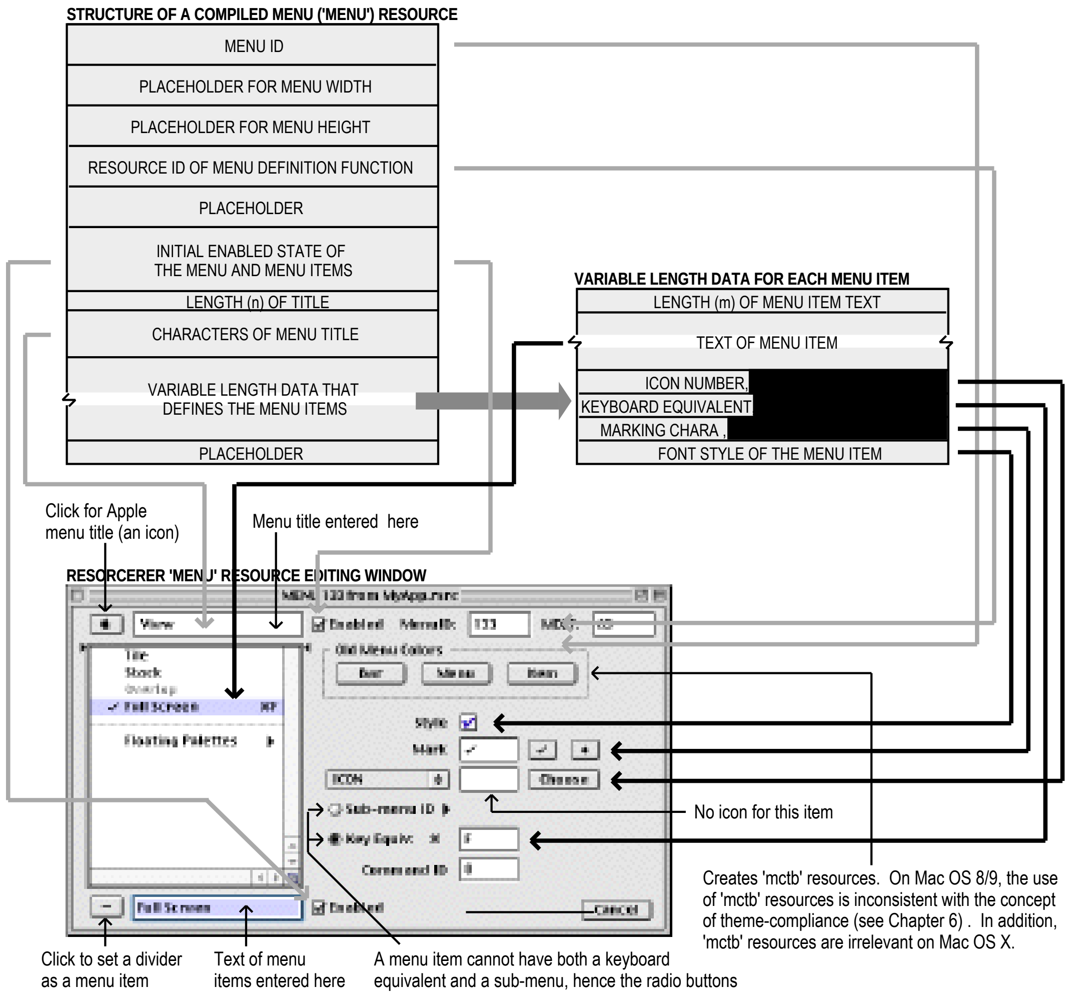

STRUCTURE OF A COMPILED MENU ('MENU') RESOURCE
MENU ID
PLACEHOLDER FOR MENU WIDTH
PLACEHOLDER FOR MENU HEIGHT
RESOURCE ID OF MENU DEFINITION FUNCTION
PLACEHOLDER
INITIAL ENABLED STATE OF THE MENU AND MENU ITEMS
LENGTH (n) OF TITLE
CHARACTERS OF MENU TITLE
VARIABLE LENGTH DATA THAT DEFINES THE MENU ITEMS
PLACEHOLDER
VARIABLE LENGTH DATA FOR EACH MENU ITEM
LENGTH (m) OF MENU ITEM TEXT
TEXT OF MENU ITEM
ICON NUMBER,
KEYBOARD EQUIVALENT
MARKING CHARA ,
FONT STYLE OF THE MENU ITEM
Click for Apple menu title (an icon)
Menu title entered here
RESORCERER 'MENU' RESOURCE EDITING WINDOW
No icon for this item
Creates 'mctb' resources. On Mac OS 8/9, the use of 'mctb' resources is inconsistent with the concept of theme-compliance (see Chapter 6) . In addition, 'mctb' resources are irrelevant on Mac OS X.
Click to set a divider as a menu item
Text of menu items entered here
A menu item cannot have both a keyboard equivalent and a sub-menu, hence the radio buttons

**FIG 9 - EDITING A 'MENU' RESOURCE USING RESORCERER**

Fig 10 shows the same **View** menu with the **Floating Palettes** menu item being edited. This item has a submenu; accordingly, the **Sub-menu ID** radio button has been clicked and the resource ID of the submenu's `'MENU'` resource has been entered. The item also has an icon provided by a `'CICN'` or `'cicn'` resource with a resource ID of 257.

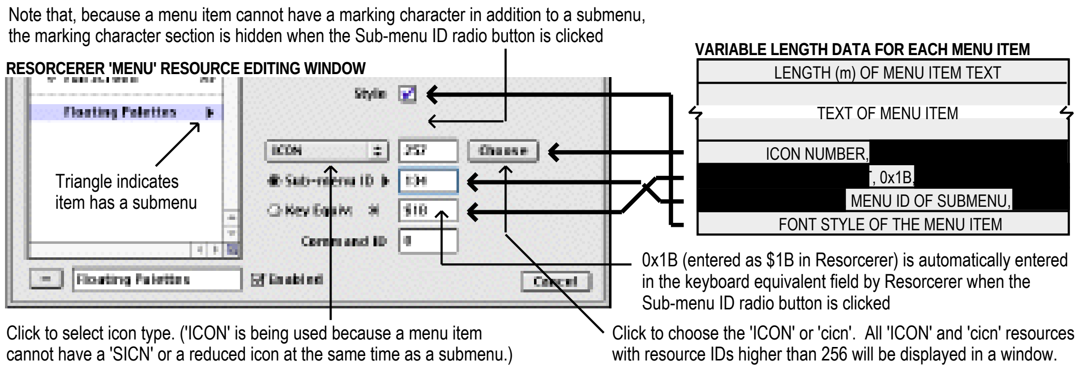

Note that, because a menu item cannot have a marking character in addition to a submenu, the marking character section is hidden when the Sub-menu ID radio button is clicked
RESORCERER 'MENU' RESOURCE EDITING WINDOW
VARIABLE LENGTH DATA FOR EACH MENU ITEM
LENGTH (m) OF MENU ITEM TEXT
TEXT OF MENU ITEM
ICON NUMBER,
, 0x1B
MENU ID OF SUBMENU,
FONT STYLE OF THE MENU ITEM
Triangle indicates item has a submenu
0x1B (entered as $1B in Resorcerer) is automatically entered in the keyboard equivalent field by Resorcerer when the Sub-menu ID radio button is clicked
Click to select icon type. ('ICON' is being used because a menu item cannot have a 'SICN' or a reduced icon at the same time as a submenu.)
Click to choose the 'ICON' or 'cicn'. All 'ICON' and 'cicn' resources with resource IDs higher than 256 will be displayed in a window.

**FIG 10 - FURTHER EDITING OF A 'MENU' RESOURCE USING RESORCERER**

## *Creating 'MENU' Resources for Submenus*

Fig 11 shows the Line and Fill submenu item in the submenu attached to the **Floating Palettes** menu item being edited. This item has a marking character (a checkmark), an icon provided by an `'ICON'` or `'cicn'` resource with resource ID 258, and a Command-key equivalent.

RESORCERER 'MENU' RESOURCE EDITING WINDOW

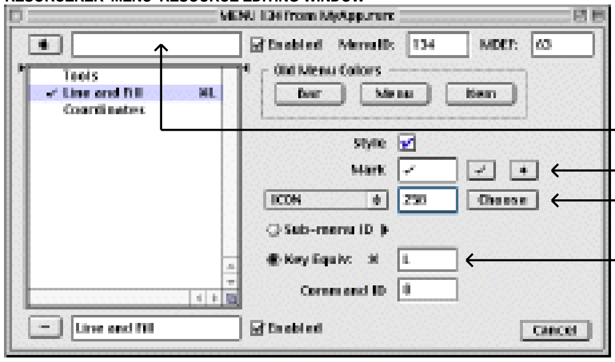
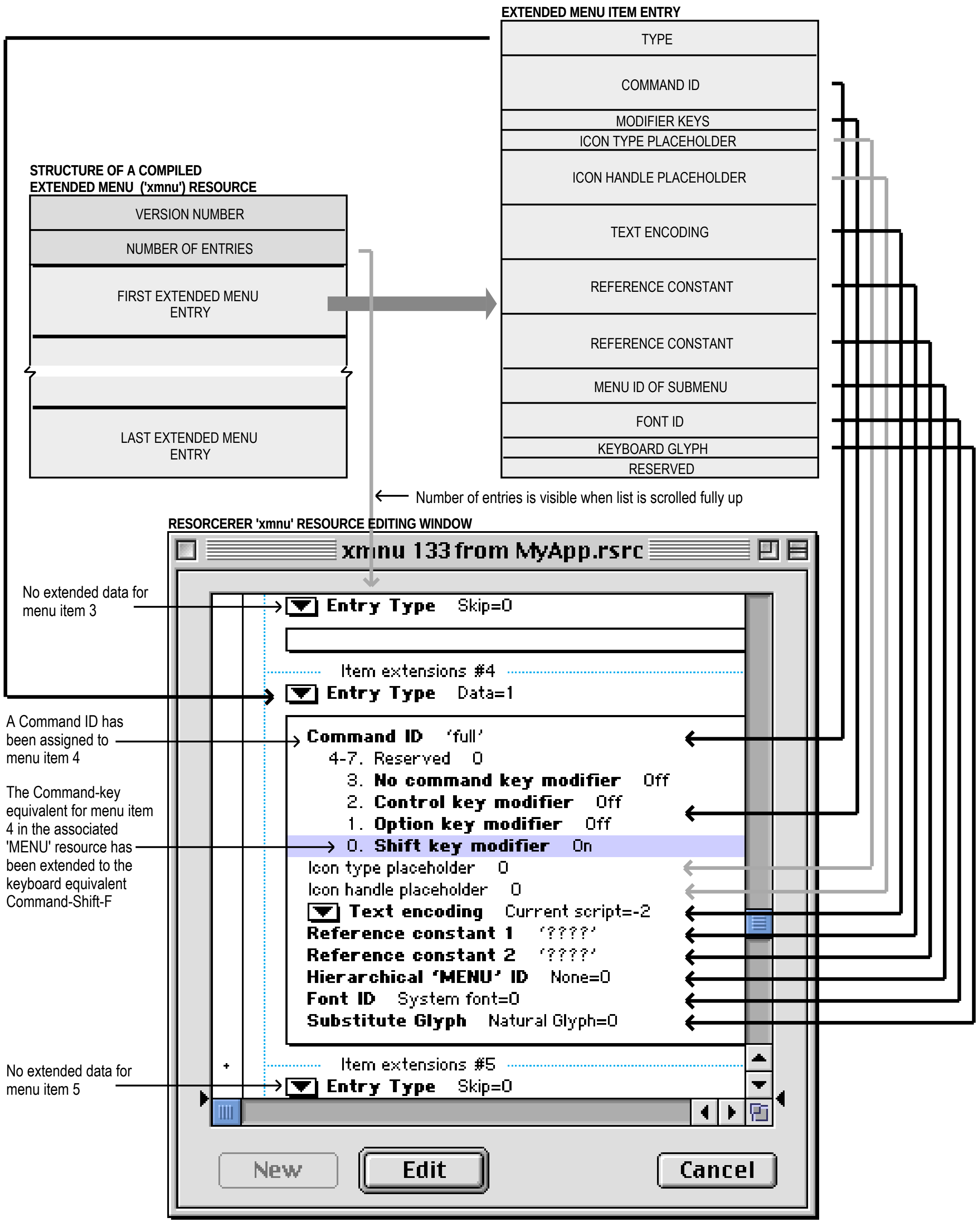


FIG 11 - EDITING A 'MENU' RESOURCE FOR A SUBMENU USING RESORCERER

## *Creating 'xmnu' Resources*

Fig 12 shows an `'xmnu'` resource being created using Resorcerer. This `'xmnu'` resource extends the `'MENU'` resource with resource ID 133 (the **View** menu, above). Menu item 4 has been assigned a command ID, and the Command-key equivalent assigned to this item in the `'MENU'` resource (Command-F) has been extended to the keyboard equivalent Command-Shift-F by specifying the Shift key as an extended modifier.


EXTENDED MENU ITEM ENTRY
TYPE
COMMAND ID
MODIFIER KEYS
ICON TYPE PLACEHOLDER
ICON HANDLE PLACEHOLDER
TEXT ENCODING
REFERENCE CONSTANT
REFERENCE CONSTANT
MENU ID OF SUBMENU
FONT ID
KEYBOARD GLYPH
RESERVED
STRUCTURE OF A COMPILED EXTENDED MENU ('xmnu') RESOURCE
VERSION NUMBER
NUMBER OF ENTRIES
FIRST EXTENDED MENU ENTRY
LAST EXTENDED MENU ENTRY
Number of entries is visible when list is scrolled fully up
RESORCERER 'xmnu' RESOURCE EDITING WINDOW
No extended data for menu item 3
A Command ID has been assigned to menu item 4
The Command-key equivalent for menu item 4 in the associated 'MENU' resource has been extended to the keyboard equivalent Command-Shift-F
No extended data for menu item 5
xmnu 133 from MyApp.rsrc
New
Edit
Cancel

```
Entry Type   Skip=0

Item extensions #4
Entry Type   Data=1

Command ID   'full'
  4-7. Reserved   0
    3. No command key modifier   Off
    2. Control key modifier   Off
    1. Option key modifier   Off
    0. Shift key modifier   On
Icon type placeholder   0
Icon handle placeholder   0
Text encoding   Current script=-2
Reference constant 1   '????'
Reference constant 2   '????'
Hierarchical 'MENU' ID   None=0
Font ID   System font=0
Substitute Glyph   Natural Glyph=0

Item extensions #5
Entry Type   Skip=0
```

FIG 12 - CREATING AN 'xmnu' RESOURCE USING RESORCERER

## *Creating the Menu Bar and Pull-Down Menus*

The function `GetNewMBar`, which itself calls `GetMenu`, should be used to read in the `'MBAR'` resource and its associated `'MENU'` resources. After reading in a `'MENU'` resource, `GetMenu` looks for an `'xmnu'` resource with the same resource ID and reads it in if found. `GetNewMBar` creates a menu object for each menu and inserts each menu into the menu list.

`SetMenuBar` should then be used to set the **current menu list** as the menu list created by your application. A call to `DrawMenuBar` completes the process by drawing the menu bar, displaying all the menu titles in the current menu list.

### *Deleting the Quit Command*

As previously stated, your application must include a **Quit** command in the **File** menu when your application is run on Mac OS 8/9 but not when it is run on Mac OS X. Accordingly, you must conditionalize your code so as to ensure that the **Quit** command and its preceding divider are deleted when your application is run on Mac OS X. The methodology recommended by Apple is as follows:

```
SInt32  response;
MenuRef menuRef;

Gestalt(gestaltMenuMgrAttr,&response);
if(response & gestaltMenuMgrAquaLayoutMask)
{
  menuRef = GetMenuRef(mFile);
  if(menuRef != NULL)
  {
    DeleteMenuItem(menuRef,iQuit);
    DeleteMenuItem(menuRef,iQuit - 1);
  }
}
```

## *Creating a Hierarchical Menu*

`GetNewMBar` does not read in the resource descriptions of submenus but simply records the menu ID of any submenu in the menu object. Submenu descriptions are read in with `GetMenu` and the submenu is inserted in the current menu list using `InsertMenu`, with the constant `hierMenu` passed as the second parameter to that call.[5]

***Carbon Note***

In Carbon, calling `GetMenu` twice on the same resource ID will create two independent, unique menus. (In Classic, the second call to `GetMenu` returns the same `MenuHandle` as the first call.) Thus, to prevent memory leaks in Carbon, `GetMenu` should not be called a second time on the same resource ID without an intervening call to `DisposeMenu`.

## *Adding Menus to the Menu List*

A menu may be added to the current menu list using one of the following procedures:

- Read the relevant `'MENU'` resource in with `GetMenu`, add it to the current menu list with `InsertMenu`, and update the menu bar with `DrawMenuBar`.
- Use `NewMenu` to create a new empty menu, use `AppendMenu`, `InsertMenuItem`, `InsertResMenu`, or `AppendResMenu` to fill the menu with menu items, add the menu to the current menu list using `InsertMenu`, and update the menu bar using `DrawMenuBar`.

Note that `GetMenuRef` may be used to obtain a reference to the menu object of any menu in the current menu list.

[5] As the user traverses menu items, if an item has a submenu, the `MenuSelect` function looks in the submenu portion of the menu list for the submenu. It then searches for a menu with a defined menu ID that matches the menu ID specified by the hierarchical menu item. If it finds a match, it attaches the submenu to the menu item.

## *Providing Help Balloons (Mac OS 8/9)*

### *'hmmu' Resources*

For Mac OS 8/9, you should define Help balloons for each of your application's menu items and each menu title. Help balloons for menus are defined in `'hmmu'` resources. The resource ID of an `'hmnu'` resource should be the same as the resource ID of the `'MENU'` resource to which it pertains.

#### *Creating 'hmnu' Resources*

Fig 13 shows an `'hmnu'` (help menu) resource being created using Resorcerer.

#### *Specifying the Format of Help Messages*

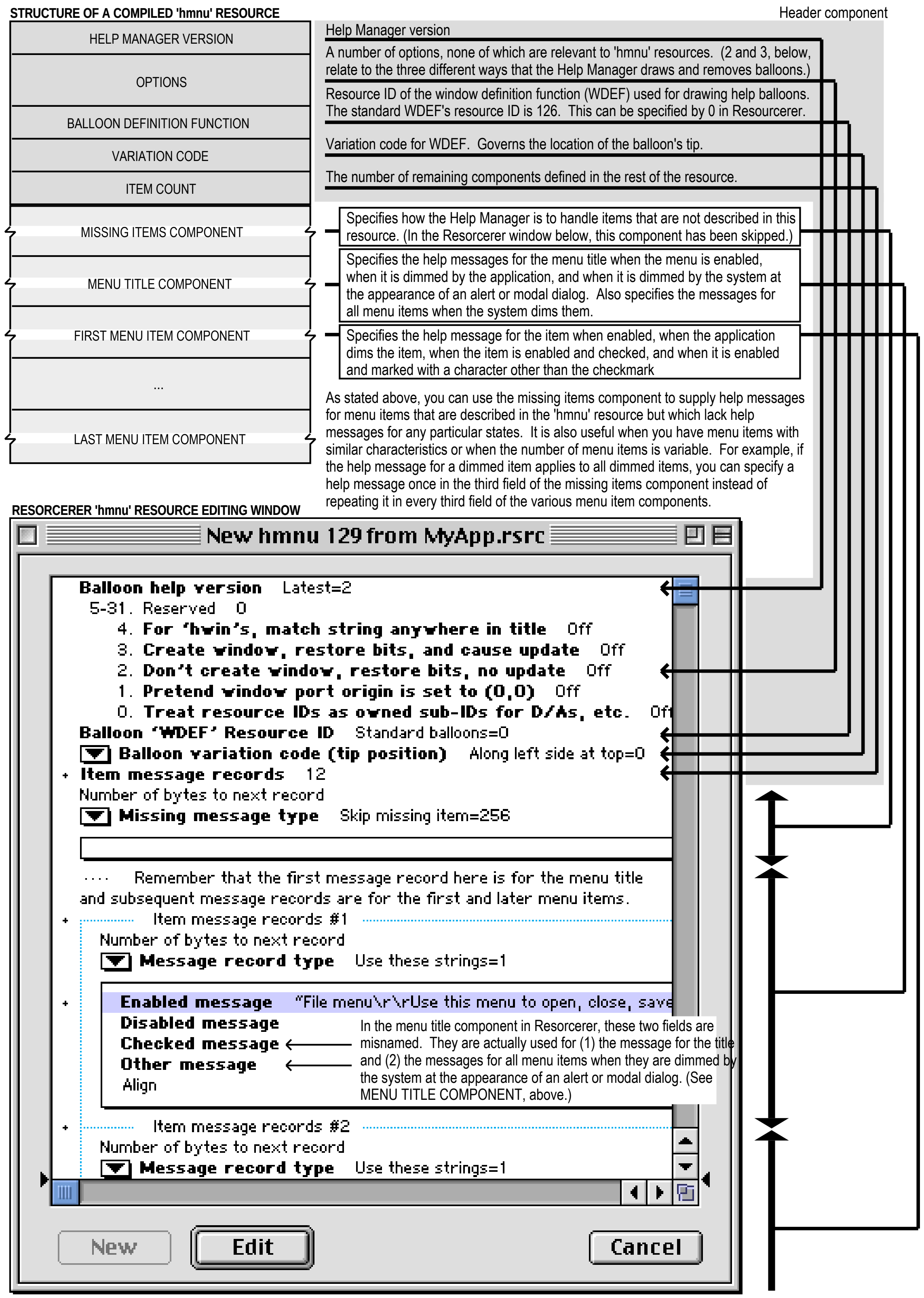
The example at Fig 13 specifies the format of the help messages as (Pascal) text strings stored within the `'hmnu'` resource itself. Clicking on the pop-up button adjacent to **Message record type** opens a pop-up menu that facilitates the choice of other formats (and also provides an option that enables you to instruct the Help Manager to skip the item). The items in the pop-up menu and their meanings are as follows:

| ***Pop-up Menu Item*** | ***Meaning to Help Manager*** |
|---|---|
| **Use these strings** | Use the strings specified within this component of this `'hmnu'` resource. |
| **Use 'PICT' resources** | Use the picture stored in the specified `'PICT'` resource. |
| **Use 'STR#' resources** | Use the specified text string stored in the specified `'STR#'` resource. (Storing the text strings in `'STR#'` resources or `'STR '` resources (see below) simplifies the task of providing foreign language versions of your application.) |
| **Used styled text resources** | Use the styled text stored in the specified `'TEXT'` and `'styl'` resources. |
| **Use 'STR ' resources** | Use the text string stored in the specified `'STR '` resource. |
| **Use named resource type** | Use the resource (`'STR '`, `'PICT'` or `'TEXT'`) whose name matches the name and state of the current menu item. |
| **Skip missing item** | No help message. Skip this item. |
| **Compare item** | Compare the specified comparison string against the current menu item in that position. If the specified string matches the name of the current menu item, display the help messages specified in the next four elements. (This is useful in the case of menu items that change names, for example **Show Hidden Text** and **Hide Hidden Text.**) |

#### *Text for Help Balloons*

The text of your help balloons for menus should answer at least one of the following questions:

- ***What is this?*** For example, when the user moves the cursor over the title of the **File** menu in the title bar, the beginning of the balloon text should be "File menu".
- ***What does this do?*** For example, when the user moves the cursor over the **Find** item in a **File** menu, the balloon text should be "Finds and selects items with the characteristics you specify" or similar.



**STRUCTURE OF A COMPILED 'hmnu' RESOURCE**

Help Manager version

A number of options, none of which are relevant to 'hmnu' resources. (2 and 3, below, relate to the three different ways that the Help Manager draws and removes balloons.)

Resource ID of the window definition function (WDEF) used for drawing help balloons. The standard WDEF's resource ID is 126. This can be specified by 0 in Resourcerer.

Variation code for WDEF. Governs the location of the balloon's tip.

The number of remaining components defined in the rest of the resource.

Specifies how the Help Manager is to handle items that are not described in this resource. (In the Resorcerer window below, this component has been skipped.)

Specifies the help messages for the menu title when the menu is enabled, when it is dimmed by the application, and when it is dimmed by the system at the appearance of an alert or modal dialog. Also specifies the messages for all menu items when the system dims them.

Specifies the help message for the item when enabled, when the application dims the item, when the item is enabled and checked, and when it is enabled and marked with a character other than the checkmark

As stated above, you can use the missing items component to supply help messages for menu items that are described in the 'hmnu' resource but which lack help messages for any particular states. It is also useful when you have menu items with similar characteristics or when the number of menu items is variable. For example, if the help message for a dimmed item applies to all dimmed items, you can specify a help message once in the third field of the missing items component instead of repeating it in every third field of the various menu item components.

**RESORCERER 'hmnu' RESOURCE EDITING WINDOW**

New hmnu 129 from MyApp.rsrc

```
Balloon help version   Latest=2
 5-31. Reserved   0
    4. For 'hwin's, match string anywhere in title   Off
    3. Create window, restore bits, and cause update   Off
    2. Don't create window, restore bits, no update   Off
    1. Pretend window port origin is set to (0,0)   Off
    0. Treat resource IDs as owned sub-IDs for D/As, etc.   Off
Balloon 'WDEF' Resource ID   Standard balloons=0
Balloon variation code (tip position)   Along left side at top=0
Item message records   12
Number of bytes to next record
Missing message type   Skip missing item=256

····   Remember that the first message record here is for the menu title
and subsequent message records are for the first and later menu items.
   Item message records #1
  Number of bytes to next record
  Message record type   Use these strings=1

   Enabled message   "File menu\r\rUse this menu to open, close, save
   Disabled message
   Checked message
   Other message
   Align
   Item message records #2
  Number of bytes to next record
  Message record type   Use these strings=1
```

In the menu title component in Resorcerer, these two fields are misnamed. They are actually used for (1) the message for the title and (2) the messages for all menu items when they are dimmed by the system at the appearance of an alert or modal dialog. (See MENU TITLE COMPONENT, above.)

FIG 13 - CREATING AN 'hmnu' RESOURCE USING RESORCERER

## Changing Menu Item Appearance

Menu Manager functions may be used to change the appearance of items in a menu, for example, the font style, text or other characteristics.

### Enabling and Disabling Menu Items

Specific menu items or entire menus may be disabled and enabled using `DisableMenuItem` and `EnableMenuItem`, which both take a reference to the menu object that identifies the desired menu and either the item number of the menu to be enabled/disabled or a value of 0 to indicate that the entire menu is to be enabled/disabled. Alternatively, if your application uses command IDs to identify menu items, you should use the functions `EnableMenuCommand` and `DisableMenuCommand` to enable and disable items.

When an entire menu is disabled or enabled, `DrawMenuBar` should be called to update the appearance of the menu bar. If you do not need to update the menu bar immediately, you can use `InvalMenuBar` instead of `DrawMenuBar`, causing the Event Manager to redraw the menu bar the next time it scans for update events. This will reduce the menu bar flicker that will occur if `DrawMenuBar` is called more than once in rapid succession.

If you disable an entire menu, the Menu Manager dims that menu's title at the next call to `DrawMenuBar` and dims all menu items 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.

#### Enabling the Preferences... Item in the Application Menu

The **Preferences...** item in the Mac OS X Application menu is disabled by default. If your application needs to allow the user to invoke a Preferences dialog by choosing this item, it must explicitly enable the item. The following shows how to enable the **Preferences...** item:

```
EnableMenuCommand(NULL,kHICommandPreferences);
```

### Other Appearance Changes

The following lists other functions related to changing the appearance of menu items.

| *Function* | *Description* |
|---|---|
| `SetMenuItemText`<br>`SetMenuItemTextWithCFString`<br>`GetMenuItemText` | Set and get the text. |
| `SetItemStyle`<br>`GetItemStyle` | Set and get the font style. |
| `SetItemMark`<br>`GetItemMark` | Set and get the marking character. |
| `SetItemIcon`<br>`GetItemIcon` | Set and get the icon (`'ICON'` or `'cicn'`) using a resource ID. |
| `CheckMenuItem` | Places and removes a checkmark at the left of the item text. |
| `SetMenuItemFontID`<br>`GetMenuItemFontID` | Set and get the font. `SetMenuItemFontID` allows you to set up a font menu with each item being drawn in the actual font. |
| `SetMenuItemIconHandle`<br>`GetMenuItemIconHandle` | Set and get the icon (icon suite, `'ICON'` or `'cicn'`) using an icon handle. Provides, in conjuction with the `'xmnu'` resource, the support for icon suites introduced with Mac OS 8. |
| `SetMenuItemKeyGlyph`<br>`GetMenuItemKeyGlyph` | `SetMenuItemKeyGlyph` substitutes a keyboard glyph for that normally displayed for a menu item's keyboard equivalent. `GetMenuItemKeyGlyph` gets the keyboard glyph for the keyboard equivalent. |
| `SetMenuFont`<br>`GetMenuFont` | Set and get the font used in an individual menu. |
| `SetMenuExcludesMarkColumn`<br>`GetMenuExcludesMarkColumn` | Set and get whether an individual menu contains space for marking characters. |

## Adding Items to a Menu

### Adding Items Other Than the Names of Resources

AppendMenu, InsertMenuItem, AppendMenuItemText, AppendMenuItemTextWithCFString, InsertMenuItemText and InsertMenuItemTextWithCFString are used to add items other than the names of resources (such as font resources) to a previously created menu. They require:

· A reference to the menu object of the menu involved.

· A string describing the items to add.

#### Strings With Metacharacters

AppendMenu and InsertMenuItem allow you to specify the same characteristics for menu items as are available when defining a 'MENU' resource. The string consists of the text of the menu item and any required characteristics. You can specify a hyphen as the menu item text to create a divider. You can also use various **metacharacters** in the text string to separate menu items and to specify the required characteristics. The following metacharacters may be used:

| *MetaCharacter* | *Description* |
|---|---|
| ; or Return | Separates menu items. |
| ^ | When followed by an icon number, defines the icon for the item. |
| ! | When followed by a character, defines the mark for the item.<br>If the keyboard equivalent field contains 0x1B, this value is interpreted as the menu ID of a submenu of this menu item.1 |
| < | When followed by one or more of the characters B, I, U, O, and S, defines the character style of the item to, respectively, bold, italic, underline, outline or shadow. |
| / | When followed by a character, defines the Command-key equivalent for the item.[2]<br>When followed by 0x1B, specifies that this menu item has a submenu.[1]<br>(Note: To specify that a menu item has a script code, reduced icon or small icon, use SetItemCmd to set the keyboard equivalent field to, respectively, 0x1C, 0x1D or 0x1E.[3] ) |
| ( | Defines the menu item as disabled. |

[1] Applicable only to menus without 'xmnu' resources. When 'xmnu' resources are used, use SetMenuItemHierarchicalID to attach a submenu to a menu item.

[2] When 'xmnu' resources are used, use SetMenuItemModifiers to set the extended modifier keys (Shift, Option, Control).

[3] Applicable only to menus without 'xmnu' resources. When 'xmnu' resources are used, do not use SetItemCmd to specify a script code. Use SetMenuItemTextEncoding.

As an example of the use of metacharacters, assume that the following two strings are stored in a string list ('STR#') resource:

```
Pick a Colour...
(^2!=Everything<B/E
```

The second string in this resource uses metacharacters to specify that the menu item is to be disabled, that it has an icon with a resource ID 258 (2+256)[6], that it has the "=" character as a marking character, that the text style is bold, and that the item has a Command-key equivalent of Command-E.

#### Examples

The following code uses AppendMenu to append a menu item with no specific characteristics other than its text to the menu identified by the menu reference. The text for the menu item is "Pick a Colour..." as stored in the preceding 'STR#' resource.

```
MenuRef menuRef;
Str255  itemString;
...
menuRef = GetMenuRef(mLibrary);
```

[6] The Menu Manager adds 256 to the number you specify, and uses the result as the icon's resource ID.

```
    GetIndString(itemString,300,1);
    AppendMenu(menuRef,itemString);
```

To insert an item after a given menu item, use `InsertMenuItem`. The following code inserts the menu item "`Everything`" after the menu item with the item number specified in the `iRed` constant:

```
    MenuRef menuRef;
    Str255  ItemString;
    ...
    menuRef = GetMenuRef(mColours);
    GetIndString(itemString,300,2);
    InsertMenuItem(menuRef,itemString,iRed);
```

The following code appends multiple items to the **Edit** menu using `AppendMenu`:

```
    MenuRef menuRef;
    ...
    menuRef = GetMenuRef(mEdit);
    AppendMenu(menuRef,"\pUndo/Z;-;Cut/X;Copy/C;Paste/V");
```

`InsertMenuItem` differs from `AppendMenu` in the way it handles the given text string when that string contains multiple items, inserting them in reverse order. This code is equivalent to the last line of the preceding code:

```
    InsertMenuItem(menuRef,"\pPaste/V;Copy/C;Cut/X-;-;Undo/Z",0);
```

The following code adds a divider to the Edit menu:

```
    AppendMenu(menuRef,"\p(-");
```

### Strings Without Metacharacters

`AppendMenuItemText`, `AppendMenuItemTextWithCFString`, `InsertMenuItemText` and `InsertMenuItemTextWithCFString` append and insert the specified string without evaluating the string for metacharacters. These functions may be used if you have a need to present non-alphanumeric characters in a menu item.

### Adding Items Comprising Resource Names to a Menu

`AppendResMenu` or `InsertResMenu` may be used to add items that consist of resource names to a menu. For example, you can use `AppendResMenu` to add the names of all resident fonts as menu items in your application's **Font** menu.

## Associating Data With Menu Items

The following functions may be used by your application to associate data with menu items:

| *Function* | *Description* |
|---|---|
| `SetMenuItemProperty` | Associates data with a menu item. |
| `GetMenuItemPropertySize` | Obtains the size of a piece of data that has been previously associated with a menu item. |
| `GetMenuItemProperty` | Obtains a piece of data that has been previously associated with a menu item. |
| `RemoveMenuItemProperty` | Removes a piece of data that has been previously associated with a menu item. |

Note that these functions require a command ID to be passed in their second parameter. Accordingly, they may only be used when your application uses command IDs to refer to the relevant menu items.

## Handling Menu Choices

### Determining the Menu ID and Menu Item — MenuSelect and MenuEvent

The first action your application should take when the user presses the mouse button while the cursor is in the menu bar is menu adjustment (that is, enabling or disabling menu items and adding or removing marks as required). Your application should then call `MenuSelect`. `MenuSelect` tracks the mouse, displays menus, highlights menu titles, displays and highlights enabled menu items, handles all user activity until the user releases the mouse button, and returns a long integer as its function result. The long integer contains the menu ID in the high word and the item number in the low word.

If some of your menu items have keyboard equivalents, your application should detect such key-down events. If an examination of the `modifiers` field of the event structure reveals that the Command key was down, your application should first adjust its menus and then call `MenuEvent`. `MenuEvent` scans the current menu list for a menu item that has a matching keyboard equivalent. Like `MenuSelect`, `MenuEvent` returns a long integer containing the menu ID and the item number.

If the user did not actually choose a menu command with the mouse, or if the user pressed a keyboard combination that did not map to a keyboard equivalent, `MenuSelect` and `MenuEvent` return 0 in the high word, the value in the low word being undefined.

#### *Further Handling - Command IDs Not Used*

The long integer returned by `MenuSelect` and `MenuEvent` should be passed as a parameter to a function that switches according to the menu ID in the high word and passes the low word to other functions that respond appropriately to that menu command.

#### *Further Handling - Command IDs Used*

Mac OS 8 introduced an alternative method of identifying, for the purposes of further handling, the menu item chosen by the user. This method assumes that you have previously assigned a unique value to your individual menu items via the command ID field of the `'xmnu'` resource (or, programmatically, via calls to `SetMenuItemCommandID`).

Using this method, the menu ID and item number should be extracted from the long integer returned by `MenuSelect` and `MenuEvent` in the usual way. The menu ID should then be used in a call to `GetMenuRef` to get the reference to the menu's menu object. This reference and the menu item should then be used in a call to `GetMenuItemCommandID`, which returns the unique value that you previously assigned to the item (that is, the item's command ID). Your application should then switch according to that command ID, calling the other functions that respond appropriately to that menu command.

### *Unhighlighting the Menu Title*

Recall that one of the actions of `MenuSelect` and `MenuEvent` is to highlight the menu title. Ordinarily, your application should not unhighlight the menu title (using `HiliteMenu`) until it performs the action associated with the menu command chosen by the user. However, if, in response to a menu command, your application displays a modal dialog containing an edit text item, you should unhighlight the menu title immediately so that the user can access the **Edit** menu.

### *Adjusting Menus*

Menu adjustment should be on the basis of the type of window that is currently the frontmost window, for example, a text window, a modeless dialog, etc. Accordingly, the menu adjustment function should first determine which window is the front window. The following are examples of menu adjustment functions:

```
void  doAdjustMenus(void)
{
  WindowRef windowRef;
  SInt16    windowType;

  windowRef = FrontWindow();
  windowType = doGetWindowType(windowRef);

  switch windowType
  {
    case kMyDocWindow:
      doAdjustFileMenuForDocWindow();
      doAdjustEditMenuForDocWindow();
      // Adjust others.
      break;
    case kMyModelessDialogWindow:
      doAdjustMenusForModelessDialogs();
      break;

    case kNil:
      doAdjustMenusNoWindows();
      break;
  };

  DrawMenuBar;
}

void doAdjustFileMenuForDocWindow(void)
{
  MenuRef menuRef;

  menuRef = GetMenuRef(mFile);

  EnableMenuItem (menuRef,iNew);
  EnableMenuItem (menuRef,iOpen);
  DisableMenuItem(menuRef,iClose);
  DisableMenuItem(menuRef,iSave);
  DisableMenuItem(menuRef,iSaveAs);
  DisableMenuItem(menuRef,iPageSetup);
  DisableMenuItem(menuRef,iPrint);
  EnableMenuItem (menuRef,iQuit);
}
```

## *Handling Mac OS 8/9 Apple Menu Choices*

When the user chooses an item in the Mac OS 8/9 Apple menu, `MenuSelect` returns the menu ID of your application's Apple menu in the high word and the item number in the low word.

If your application provides an **About** command as the first menu item in the Apple menu, and the user chooses this item, you should display the About dialog/alert. Other choices from the Mac OS 8/9 Apple menu are handled automatically by the system.

## *Handling Mac OS X Application Menu Choices*

### *About Command*

When the user chooses the **About** command in the Mac OS X Application menu, `MenuSelect` returns the menu ID of your application's Application menu in the high word and 1 in the low word. Thus the code that handles the user's choice of the Apple menu's **About** item on Mac OS 8/9 will also handle the user's choice of the Application menu's **About** item on Mac OS X.

### *Quit Command*

When the user chooses the **Quit** command from the Mac OS X Application menu, a high-level event (more specifically, an Apple event) known as a Quit Application event is sent to your application. Your application must support the Quit Application event in order to respond to the user choosing the **Quit** command. (See Chapter 10.)

### *Preferences… Command*

An Apple Event, known as the Show Preferences event, is sent to your application when the user chooses the **Preferences…** command from the Mac OS X Application menu. Your application must support the Show Preferences event in order to respond to the user choosing the **Preferences…**command. (See the demonstration program at Chapter 19.)

## Handling a Size Menu

### Preamble

On Mac OS 8/9, font sizes in **Size** menus should be outlined to indicate which sizes are directly provided by the current font. For bitmapped fonts, you should outline only those sizes that exist in the Fonts folder. For TrueType fonts, all sizes supported by that font should be outlined. The current font size should be indicated with a checkmark. If the current selection contains more than one font size, a dash should be placed next to each font size in the selection.

**Size** menus should, in addition to displaying available font sizes, provide an **Other** command to enable the user to specify a size not currently listed in the menu. When the user chooses the **Other** command, the current font size should be displayed in a dialog which allows the user to enter the desired font size. If the user chooses a size not already in the menu, a checkmark should be added to the **Other** menu item and the chosen size should be added in parenthesis to the text of the **Other** command.

### Handling the Menu Choice

The following is an example function that handles a user's choice of an item in a Size menu:

```
void  doHandleSizeCommand(SInt16 menuItem)
{
  SInt16  numItems;
  Boolean addItem;
  SInt32  sizeChosen;

  numItems = CountMenuItems(GetMenuHandle(mSize));
  if(menuItem == numItems)          // If user chose Other, display dialog. If the
  {                                 // user-specified size is not in the menu, add a
    doDisplayOtherBox(sizeChosen);  // checkmark to the Other command and add the new
  }                                 // font size to the text of the Other command.
  else                              // Return sizeChosen.
  {                                 // User chose a size.
    doRemoveMarksFromSizeMenu();    // Remove marks from item/s showing previous size.
    CheckMenuItem(GetMenuHandle(mSize),menuItem,true);  // Add mark to chosen item.
    sizeChosen = doItemToSize(menuItem);     // Convert item number to font size.
  }
  doResizeSelection(sizeChosen);             // Update document state or user selection.
}
```

## Hiding and Showing the Menu Bar

`HideMenuBar` and `ShowMenuBar` may be used to make the menu bar invisible and unselectable and visible and selectable. On Mac OS X, these functions also hide and show the Dock.

## Accessing Menus From Alerts and Dialogs

When alerts and dialogs are displayed, the Dialog Manager and the Menu Manager interact to provide varying degrees of access to menus in your menu bar. In some circumstances, you can rely on the system software to disable the appropriate menus and menu items. In other circumstances, you application must contribute to, or control, the matter of menu access.

The subject of menu access when alerts, movable alerts, modal dialogs, moveable modal dialogs, and modeless dialogs are displayed is somewhat involved, and is addressed in detail at Chapter 8.

## Main Menu Manager Constants, Data Types, and Functions

## Constants

### For markChar Parameter of SetItemMark Calls

```
noMark       = 0
commandMark  = 17
checkMark    = 18
diamondMark  = 19
appleMark    = 20
```

### For beforeID Parameter of InsertMenu to Insert a Submenu Into the Submenu Portion of the Menu List

```
hierMenu     = -1
```

### For options Parameter of CreateStandardFontMenu Calls

```
KHierarchicalFontMenuOption  = 0x00000001
```

### Modifier Key Masks for GetMenuItemModifiers and SetMenuItemModifiers Calls

```
kMenuCommandModifiers  = 0         If no bit is set, only the Command key is used.
kMenuShiftModifier     = (1 << 0)  If this bit (bit 0) is set, the Shift key is used.
kMenuOptionModifier    = (1 << 1)  If this bit (bit 1) is set, the Option key is used.
kMenuControlModifier   = (1 << 2)  If this bit (bit 2) is set, the Control key is used.
kMenuNoCommandModifier = (1 << 3)  If this bit (bit 3) is set, the Command key is not used.
```

### Menu Icon Handle Constants for GetMenuItemIconHandle and SetMenuItemIconHandle Calls

```
kMenuNoIcon            = 0  No icon.
kMenuIconType          = 1  'ICON' handle.
kMenuShrinkIconType    = 2  32-by-32 'ICON' handle shrunk (at display time) to 16-by-16.
kMenuSmallIconType     = 3  'SICN' handle.
kMenuColorIconType     = 4  'cicn' handle.
kMenuIconSuiteType     = 5  Icon suite handle.
```

### Menu Attributes

```
kMenuAttrExcludesMarkColumn = (1 << 0)
kMenuAttrAutoDisable        = (1 << 2)
```

## Data Types

```
typedef struct OpaqueMenuHandle*  MenuHandle;
typedef MenuHandle  MenuRef;
typedef SInt16      MenuID;
typedef UInt16      MenuItemIndex;
typedef UInt32      MenuCommand;
typedef Handle      MenuBarHandle;
typedef UInt32      MenuAttributes;
```

## Functions

### Creating and Disposing Of Menus

```
MenuRef   NewMenu(MenuID menuID, ConstStr255Param menuTitle);
MenuRef   GetMenu(short resourceID);
void      DisposeMenu(MenuRef theMenu);
```

### Creating a Help Menu (Mac OS X)

```
OSStatus  HMGetHelpMenu(MenuRef *outHelpMenu,MenuItemIndex *outFirstCustomItemIndex);
```

### Adding Menus to and Removing Menus From the Current Menu List

```
void      InsertMenu(MenuRef theMenu,MenuID beforeID);
void      DeleteMenu(MenuID menuID);
void      ClearMenuBar(void);
```

### Getting a MenuBar Description From an 'MBAR' resource

```
MenuBarHandle  GetNewMBar(short menuBarID);
```

### Getting, Setting and Disposing of the Menu Bar

```
MenuBarHandle  GetMenuBar(void);
void           SetMenuBar(MenuBarHandle mbar);
OSStatus       DisposeMenuBar(MenuBarHandle mbar);
short          GetMBarHeight(void);
```

### Drawing the Menu Bar

```
void      DrawMenuBar(void);
void      InvalMenuBar(void);
```

### Controlling Menu Bar Visibility

```
void      HideMenuBar(void);
void      ShowMenuBar(void);
Boolean   IsMenuBarVisible(void);
```

### Modifying the Menu Width

```
Boolean   GetMenuExcludesMarkColumn(MenuRef menu);
OSStatus  SetMenuExcludesMarkColumn(MenuRef menu,Boolean excludesMark);
```

### Responding to User Choice of a Menu Command

```
Uint32    MenuEvent(const EventRecord *inEvent);
long      MenuSelect(Point startPt);
long      MenuChoice(void);
void      HiliteMenu(MenuID menuID);
long      PopUpMenuSelect(MenuRef menu,short top,short left,short popUpItem);
```

### Getting a Reference to a Menu object

```
MenuRef   GetMenuRef(MenuID menuID);
```

### Adding and Deleting Menu Items

```
void      AppendMenu(MenuRef menu,ConstStr255Param data);
void      InsertMenuItem(MenuRef theMenu,ConstStr255Param itemString,short afterItem);
OSStatus  AppendMenuItemText(MenuRef menu,ConstStr255Param inString);
OSStatus  AppendMenuItemTextWithCFString(MenuRef menu,CFStringRef inString,
          MenuItemAttributes inAttributes,MenuCommand inCommandID,MenuItemIndex *outNewItem);
OSStatus  InsertMenuItemText(MenuRef menu,ConstStr255Param inString,UInt16 afterItem);
OSStatus  InsertMenuItemTextWithCFString(MenuRef menu,CFStringRef inString,
          MenuItemIndex inAfterItem,MenuItemAttributes inAttributes,MenuCommand inCommandID);
void      DeleteMenuItem(MenuRef theMenu,short item);
void      AppendResMenu(MenuRef theMenu,ResType theType);
void      InsertResMenu(MenuRef theMenu,ResType theType,short afterItem);
```

### Enabling and Disabling Menus, Menu Items, and Menu Item Icons

```
void      EnableMenuItem(MenuRef theMenu,MenuItemIndex item);
void      DisableMenuItem(MenuRef theMenu,MenuItemIndex item);
Boolean   IsMenuItemEnabled(MenuRef menu,MenuItemIndex item);
void      DisableAllMenuItems(MenuRef theMenu);
void      EnableAllMenuItems(MenuRef theMenu);
Boolean   MenuHasEnabledItems(MenuRef theMenu);
void      EnableMenuCommand(MenuRef theMenu,MenuCommand commandID);
void      DisableMenuCommand(MenuRef theMenu,MenuCommand commandID);
Boolean   IsMenuCommandEnabled(MenuRef menu,MenuCommand commandID);
void      EnableMenuItemIcon(MenuRef theMenu,MenuItemIndex item);
void      DisableMenuItemIcon(MenuRef theMenu,MenuItemIndex item);
Boolean   IsMenuItemIconEnabled(MenuRef menu,MenuItemIndex item);
```

### Getting and Setting Menu Item Command IDs

```
OSErr     GetMenuItemCommandID(MenuRef inMenu,SInt16 inItem,UInt32 *outCommandID);
OSErr     SetMenuItemCommandID(MenuRef inMenu,SInt16 inItem,UInt32 inCommandID);
```

### Menu Object Accessor Functions

```
MenuID    GetMenuID(MenuRef menu);
void      SetMenuID(MenuRef menu,MenuID menuID);
SInt16    GetMenuWidth(MenuRef menu);
void      SetMenuWidth(MenuRef menu,SInt16 width);
SInt16    GetMenuHeight(MenuRef menu);
void      SetMenuHeight(MenuRef menu,SInt16 height);
```

```
StringPtr GetMenuTitle(MenuRef menu,Str255 title);
OSStatus  SetMenuTitle(MenuRef menu,ConstStr255Param title);
OSStatus  SetMenuTitleWithCFString(MenuRef menu,CFStringRef inString);
OSStatus  GetMenuDefinition(MenuRef menu,MenuDefSpecPtr outDefSpec);
OSStatus  SetMenuDefinition (MenuRef menu,const MenuDefSpec *defSpec);
```

### *Manipulating and Accessing Menu Item Characteristics*

```
void      GetMenuItemText(MenuRef menu,short item,Str255 itemString);
void      SetMenuItemText(MenuRef theMenu, short item, ConstStr255Param itemString);
OSStatus  SetMenuItemTextWithCFString(MenuRef menu,MenuItemIndex item,CFStringRef inString);
void      GetItemStyle(MenuRef theMenu,short item,Style* chStyle);
void      SetItemStyle(MenuRef theMenu,short item, StyleParameter chStyle);
void      GetItemMark(MenuRef theMenu,short item,CharParameter *markChar);
void      SetItemMark(MenuRef theMenu,short item,CharParameter markChar);
void      CheckMenuItem(MenuRef theMenu,short item,Boolean checked);
OSStatus  GetMenuFont(MenuRef menu,SInt16 *outFontID,UInt16 *outFontSize);
OSStatus  SetMenuFont(MenuRef menu,SInt16 inFontID,UInt16 inFontSize);
void      GetItemIcon(MenuRef theMenu,short item,short *iconIndex);
void      SetItemIcon(MenuRef theMenu,short item,short iconIndex);
void      GetItemCmd(MenuRef theMenu,short item,short *cmdChar);
void      SetItemCmd(MenuRef theMenu,short item,short cmdChar);
OSErr     GetMenuItemFontID(MenuRef inMenu,SInt16 inItem,SInt16* outFontID);
OSErr     SetMenuItemFontID(MenuRef inMenu,SInt16 inItem,SInt16 inFontID);
OSErr     GetMenuItemHierarchicalID(MenuRef inMenu,SInt16 inItem,SInt16 *outHierID);
OSErr     SetMenuItemHierarchicalID(MenuRef inMenu,SInt16 inItem,SInt16 inHierID);
OSErr     GetMenuItemIconHandle(MenuRef inMenu,SInt16 inItem,MenuIconType outIconType,
          Handle* outIconHandle);
OSErr     SetMenuItemIconHandle(MenuRef inMenu,SInt16 inItem,MenuIconType inIconType,
          Handle inIconHandle);
OSErr     GetMenuItemKeyGlyph(MenuRef inMenu,SInt16 inItem,SInt16 *outGlyph);
OSErr     SetMenuItemKeyGlyph(MenuRef inMenu,SInt16 inItem,SInt16 inGlyph);
OSErr     GetMenuItemModifiers(MenuRef inMenu,SInt16 inItem,SInt16* outModifiers);
OSErr     SetMenuItemModifiers(MenuRef inMenu,SInt16 inItem,SInt16 inModifiers);
OSErr     GetMenuItemRefCon(MenuRef inMenu,SInt16 inItem,SInt32* outRefCon);
OSErr     SetMenuItemRefCon(MenuRef inMenu,SInt16 inItem,SInt32 inRefCon);
OSErr     GetMenuItemTextEncoding(MenuRef inMenu,SInt16 inItem,TextEncoding* outScriptID);
OSErr     SetMenuItemTextEncoding(MenuRef inMenu,SInt16 inItem,TextEncoding inScriptID);
```

### *Getting a Menu Reference and Menu ID from a Command ID*

```
OSStatus  GetIndMenuItemWithCommandID(MenuRef menu,MenuCommand commandID,UInt32 itemIndex,
          MenuRef *outMenu,MenuItemIndex *outIndex);
```

### *Associating Data With Menu Items*

```
OSStatus  SetMenuCommandProperty(MenuRef menu,MenuCommand commandID,OSType propertyCreator,
          OSType propertyTag,ByteCount propertySize,const void *propertyData);
OSStatus  GetMenuCommandPropertySize(MenuRef menu,MenuCommand commandID,OSType propertyCreator,
          OSType propertyTag,ByteCount *size);
OSStatus  GetMenuCommandProperty(MenuRef menu,MenuCommand commandID,OSType propertyCreator,
          OSType propertyTag,ByteCount bufferSize,ByteCount *actualSize,void *propertyBuffer);
OSStatus  RemoveMenuCommandProperty(MenuRef menu,MenuCommand commandID,OSType propertyCreator,
          OSType propertyTag);
```

### *Counting Items in a Menu*

```
short     CountMenuItems(MenuRef theMenu);
ItemCount CountMenuItemsWithCommandID(MenuRef menu,MenuCommand commandID);
```

### *Building and Updating Font Menus and Obtaining Font Family and Style*

```
OSStatus  CreateStandardFontMenu(MenuRef menu,MenuItemIndex afterItem,MenuID firstHierMenuID,
          OptionBits options,ItemCount *outHierMenuCount);
OSStatus  UpdateStandardFontMenu(MenuRef menu,ItemCount *outHierMenuCount);
OSStatus  GetFontFamilyFromMenuSelection(MenuRef menu,MenuItemIndex item,
          FMFontFamily *outFontFamily,FMFontStyle *outStyle);
```

### *Recalculating Menu Dimensions*

```
void      CalcMenuSize(MenuRef theMenu);
```

### *Highlighting the Menu Bar*

```
void      FlashMenuBar(MenuID menuID);
```

## Demonstration Program Menus1 Listing

```
// *******************************************************************************************
// Menus1.c                                                                CLASSIC EVENT MODEL
// *******************************************************************************************
//
// This program:
//
// •  Opens a window.
//
// •  Creates these pull-down menus: Apple, File, Edit, Font, Size, Special, and Window.
//
// •  Displays text in the window indicating the menu selection made by the user.
//
// The Apple menu includes an "About…" menu item for the program.
//
// The second menu item in the Special menu contains a submenu.
//
// The Font and Window menus are created programmatically using the functions
// CreateStandardFontMenu and CreateStandardWindowMenu.
//
// The implementation of the Size menu is nominal only.  The current size is indicated with a
// checkmark; however, the number of sizes shown is not font-dependent and there is no "Other"
// item.
//
// Because the primary purpose of the program is to demonstrate menu creation and handling, no
// code is included to update and activate/deactivate the window or to respond to events which
// are not relevant to the demonstration.
//
// The program is terminated by selecting Quit from the File menu, by pressing the keyboard
// equivalent for that item (Command-Q), or by clicking in the window's go-away box/close
// button.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  An 'MBAR' resource (preload, non-purgeable).
//
// •  'MENU' resources for the Apple, File, Edit, Font, Size, and Special drop-down menus
//    and the submenu (all preload, all non-purgeable).
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *******************************************************************************************

// ............................................................................... includes

#include <Carbon.h>

// ................................................................................ defines

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iQuit            12
#define mEdit             130
#define  iUndo            1
#define  iCut             3
#define  iCopy            4
#define  iPaste           5
#define  iClear           6
#define mFont             131
```

```
#define mSize             132
#define  iTen             1
#define  iTwelve          2
#define  iEighteen        3
#define  iTwentyFour      4
#define mSpecial          133
#define  iFirstItem       1
#define  iSecondItem      2
#define mWindow           134
#define mSubmenu          135
#define mFirstFontSubMenu 136
#define  iFirstSubItem    1
#define  iSecondSubItem   2
#define rWindowResource   128

// .......................................................................... global variables

Boolean       gDone;
ItemCount     gFontMenuHierMenuCount;
MenuItemIndex gCurrentFontSizeItem = 2;
MenuItemIndex gCurrentFontMenuItem;
MenuItemIndex gCurrentFontSubMenuItem;
MenuRef       gCurrentFontSubMenuRef = NULL;

// ....................................................................... function prototypes

void  main                   (void);
void  doPreliminaries        (void);
OSErr quitAppEventHandler    (AppleEvent *,AppleEvent *,SInt32);
void  doGetMenus             (void);
void  doEvents               (EventRecord *);
void  doMouseDown            (EventRecord *);
void  doAdjustMenus          (void);
void  doMenuChoice           (SInt32);
void  doAppleApplicationMenu (MenuItemIndex);
void  doFileMenu             (MenuItemIndex);
void  doEditMenu             (MenuItemIndex);
void  doFontMenu             (MenuID,MenuItemIndex);
void  doSizeMenu             (MenuItemIndex);
void  doSpecialMenu          (MenuItemIndex);
void  doSubMenus             (MenuItemIndex);
void  drawItemString         (Str255);

// ************************************************************************************ main

void  main(void)
{
  EventRecord eventStructure;
  WindowRef   windowRef;

  // ...................................................................... do preliminaries

  doPreliminaries();

  // ........................................................................... open a window

  if(!(windowRef = GetNewCWindow(rWindowResource,NULL,(WindowRef) -1)))
  {
    SysBeep(10);
    ExitToShell();
  }

  SetPortWindowPort(windowRef);

  // ................................................ set up menu bar and menus, then show window

  doGetMenus();
  ShowWindow(windowRef);
  // .......................................................................................................... event loop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,180,NULL))
      doEvents(&eventStructure);
  }
}

// ************************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(640);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                          NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                          0L,false);
  if(osError != noErr)
    ExitToShell();
}

// *************************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                            &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ******************************************************************************** doGetMenus

void  doGetMenus(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  OSStatus      osError;
  Str255        smallSystemFontName, itemString;
  SInt16        numberOfItems,a;

  // ........................................................................ get and set, and menu bar

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);

  // ........... if running on Mac OS X, delete Quit item and preceding divider from File menu
  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }
  }

  // ............................................. create hirearchical Font menu, checkmark small system font

  menuRef = GetMenuRef(mFont);
  if(menuRef != NULL)
  {
    osError = CreateStandardFontMenu(menuRef,0,mFirstFontSubMenu,kHierarchicalFontMenuOption,
                                    &gFontMenuHierMenuCount);
    if(osError != noErr)
      ExitToShell();
  }
  else
    ExitToShell();

   GetFontName(kThemeSmallSystemFont,smallSystemFontName);
  numberOfItems = CountMenuItems(menuRef);
  for(a=1;a<numberOfItems + 1;a++)
  {
    GetMenuItemText(menuRef,a,itemString);
    if(EqualString(itemString,smallSystemFontName,false,false))
    {
      CheckMenuItem(menuRef,a,true);
      gCurrentFontMenuItem = a;
      break;
    }
  }

  // ................................................................ create Window menu and insert into menu list

  CreateStandardWindowMenu(0,&menuRef);
  SetMenuID(menuRef,mWindow);
  InsertMenu(menuRef,0);

  // ..................................................................... get submenus and insert into window list

  menuRef = GetMenu(mSubmenu);
  if(menuRef != NULL)
    InsertMenu(menuRef,hierMenu);
  else
    ExitToShell();

  // ................................... set initial font size and checkmark associated item in Size menu

  doSizeMenu(gCurrentFontSizeItem);

  // ............................................................................................................. draw menu bar

  DrawMenuBar();
 }

// ********************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      break;

    case updateEvt:
      BeginUpdate((WindowRef) eventStrucPtr->message);
      EndUpdate((WindowRef) eventStrucPtr->message);
      break;
  }
}

// ****************************************************************************** doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;
  SInt32         menuChoice;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doAdjustMenus();
      menuChoice = MenuSelect(eventStrucPtr->where);
      doMenuChoice(menuChoice);
      break;

    case inContent:
      if(windowRef != FrontWindow())
        SelectWindow(windowRef);
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where))
        gDone = true;
      break;
  }
}

// **************************************************************************** doAdjustMenus

void  doAdjustMenus(void)
{
  // Adjust menus here.
}

// ***************************************************************************** doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  menuID    = HiWord(menuChoice);
  menuItem  = LoWord(menuChoice);

  if(menuID == 0)
    return;

  if(menuID == mFont || ((menuID >= mFirstFontSubMenu) &&
                         (menuID <= mFirstFontSubMenu + gFontMenuHierMenuCount)))
    doFontMenu(menuID,menuItem);
  else
  {
    switch(menuID)
    {
      case mAppleApplication:
        doAppleApplicationMenu(menuItem);
        break;

      case mFile:
        doFileMenu(menuItem);
        break;

      case mEdit:
        doEditMenu(menuItem);
        break;

      case mSize:
        doSizeMenu(menuItem);
        break;

      case mSpecial:
        doSpecialMenu(menuItem);
        break;

      case mSubmenu:
        doSubMenus(menuItem);
        break;
    }
  }

  HiliteMenu(0);
}

// ****************************************************************** doAppleApplicationMenu

void  doAppleApplicationMenu(MenuItemIndex menuItem)
{
  if(menuItem == iAbout)
    drawItemString("\pAbout Menus1");
}

// ****************************************************************************** doFileMenu

void  doFileMenu(MenuItemIndex menuItem)
{
  if(menuItem  == iQuit)
    gDone = true;
}

// ****************************************************************************** doEditMenu

void  doEditMenu(MenuItemIndex menuItem)
{
  switch(menuItem)
  {
    case iUndo:
      drawItemString("\pUndo");
      break;
    case iCut:
      drawItemString("\pCut");
      break;

    case iCopy:
      drawItemString("\pCopy");
      break;

    case iPaste:
      drawItemString("\pPaste");
      break;

    case iClear:
      drawItemString("\pClear");
      break;
  }
}

// ********************************************************************************* doFontMenu

void  doFontMenu(MenuID menuID,MenuItemIndex menuItem)
{
  MenuRef      menuRef, fontMenuRef;
  OSStatus     osError;
  FMFontFamily currentFontFamilyReference;
  FMFontStyle  currentFontStyle;
  Str255       fontName, styleName, itemName;
  SInt16       a, numberOfFontMenuItems;

  menuRef = GetMenuRef(menuID);

  osError = GetFontFamilyFromMenuSelection(menuRef,menuItem,&currentFontFamilyReference,
                                           &currentFontStyle);
  if(osError == noErr)
  {
    TextFont(currentFontFamilyReference);
    TextFace(currentFontStyle);

    GetFontName(currentFontFamilyReference,fontName);
    drawItemString(fontName);

    if(menuID == mFont)
    {
      CheckMenuItem(menuRef,gCurrentFontMenuItem,false);
      gCurrentFontMenuItem = menuItem;
      CheckMenuItem(menuRef,gCurrentFontMenuItem,true);

      if(gCurrentFontSubMenuRef != NULL)
        CheckMenuItem(gCurrentFontSubMenuRef,gCurrentFontSubMenuItem,false);
    }
    else
    {
      if(gCurrentFontSubMenuRef != NULL)
        CheckMenuItem(gCurrentFontSubMenuRef,gCurrentFontSubMenuItem,false);
      gCurrentFontSubMenuRef = menuRef;
      gCurrentFontSubMenuItem = menuItem;
      CheckMenuItem(gCurrentFontSubMenuRef,gCurrentFontSubMenuItem,true);

      fontMenuRef = GetMenuRef(mFont);
      CheckMenuItem(fontMenuRef,gCurrentFontMenuItem,false);

      numberOfFontMenuItems = CountMenuItems(fontMenuRef);

      for(a=1;a<=numberOfFontMenuItems;a++)
      {
        GetMenuItemText(fontMenuRef,a,itemName);
        if(EqualString(fontName,itemName,true,true))
        {
          gCurrentFontMenuItem = a;
          break;
        }
      }

      SetItemMark(fontMenuRef,gCurrentFontMenuItem,'-');

      GetMenuItemText(menuRef,menuItem,styleName);
      DrawString("\p ");
      DrawString(styleName);
    }
  }
  else
    ExitToShell();
}

// ******************************************************************************** doSizeMenu

void  doSizeMenu(MenuItemIndex menuItem)
{
  MenuRef sizeMenuRef;

  switch(menuItem)
  {
    case iTen:
      TextSize(10);
      break;

    case iTwelve:
      TextSize(12);
      break;

    case iEighteen:
      TextSize(18);
      break;

    case iTwentyFour:
      TextSize(24);
      break;
  }

  sizeMenuRef = GetMenuRef(mSize);

  CheckMenuItem(sizeMenuRef,gCurrentFontSizeItem,false);
  CheckMenuItem(sizeMenuRef,menuItem,true);

  gCurrentFontSizeItem = menuItem;

  drawItemString("\pSize change");
}

// ***************************************************************************** doSpecialMenu

void  doSpecialMenu(MenuItemIndex menuItem)
{
  if(menuItem == iFirstItem)
    drawItemString("\pFirst Item");
}

// ******************************************************************************** doSubMenus

void  doSubMenus(MenuItemIndex menuItem)
{
  switch(menuItem)
  {
    case iFirstSubItem:
      drawItemString("\pSubitem 1");
      break;

    case iSecondSubItem:
      drawItemString("\pSubitem 2");
      break;
  }
}

// *************************************************************************** drawItemString

void  drawItemString(Str255 eventString)
{
  RgnHandle tempRegion;
  WindowRef windowRef;
  Rect      scrollBox;

  windowRef = FrontWindow();
  tempRegion = NewRgn();

  GetWindowPortBounds(windowRef,&scrollBox);

  ScrollRect(&scrollBox,0,-24,tempRegion);
  DisposeRgn(tempRegion);

  MoveTo(8,286);
  DrawString(eventString);
}

// ********************************************************************************************
```

## Demonstration Program Menus1 Comments

When this program is run, the user should choose items from all menus, including the Apple menu. Selections should be made using the mouse and, where appropriate, the Command key equivalents. The user should also note the effects on the menu bar of clicking outside, then inside, the program's window, that is, of sending the program to the background and returning it to the foreground.

### defines

Constants are established for the pull-down and submenu menu IDs and associated resource IDs, menu item numbers and subitem numbers.

The Menu Manager function CreateStandardFontMenu will be used to create a hierarchical Font menu and mFirstFontSubMenu establishes the ID of the first Font menu submenu to be created.

The last line establishes a constant for the resource ID of the 'WIND' resource.

### Global Variables

gFontMenuHierMenuCount will be assigned a value representing the number of submenus created by the Menu Manager function CreateStandardFontMenu.

GCurrentFontSizeItem will be assigned the menu item number of the chosen font size.

gCurrentFontMenuItem and gCurrentFontSubMenuItem will be used in the Font menu handling function to specify which menu and submenu items are to have a checkmark added or cleared. gCurrentFontSubMenuRef will be assigned a reference to the menu object for the currently chosen Font menu submenu.

### main

The main() function creates a window and makes its graphics port the current port, calls doGetMenus to set up the menus, shows the window and enters the main event loop.

### doPreliminaries

The large number of master pointers created by MoreMasterPointers in this program allows for the likely creation of a large number of submenus by the Menu Manager function CreateStandardFontMenu.

When the program is run on Mac OS X, the Quit item will be in the Application menu. The system informs the program of the user's choice of this item via a high-level event known as an Apple event, more specifically, an Apple event known as the Quit Application event. The call to AEInstallEventHandler installs quitAppEventHandler as the handler for this high-level event. (Apple events and Apple event handlers are explained at Chapter 10.)

### quitAppEventHandler

quitAppEventHandler is the handler for the Quit Application event installed in doPreliminaries. Basically, it sets the global variable gDone to true, which causes the program to terminate from the main event loop.

### doGetMenus

doGetMenus sets up the menu bar and the various menus.

At the first block, GetNewMBar reads in the 'MENU' resources for each menu specified in the 'MBAR' resource and creates a menu object for each of those menus. (Note that the error handling here and in other areas of this program is somewhat rudimentary: the program simply terminates.) The call to SetMenuBar makes the newly created menu list the current list.

The call to Gestalt determines whether the application is running on Mac OS 8/9 or Mac OS X. If the application is running on Mac OS X, GetMenuRef is called to get a reference to the menu object for the File menu and DeleteMenuItem is called to delete the Quit item and its preceding divider from the menu.

The third block utilizes the relatively new Menu Manager function CreateStandardFontMenu to create a hierarchical font menu. A reference to the Font menu object is passed in the first parameter. The third parameter specifies the menu ID for the first submenu to be created. The fourth parameter specifies that the Font menu be created as a hierarchical menu. The fifth parameter receives a value representing the number of submenus created. CreateStandardFontMenu itself inserts these submenus into the submenu portion of the menu list.

The fourth block checkmarks the Font menu item containing the name of the small system font. GetFontName gets the name of the small system font and CountMenuItems counts the number of items in the Font menu.

The for loop then walks the items in the Font menu looking for a match. When it finds a match, CheckMenuItem is called to checkmark the item, the global variable which keeps track of the currently selected font is assigned the number of that item, and the for loop is exited.

The fifth block creates the Window menu using the Window Manager function CreateStandardWindowMenu. The accessor function SetMenuID sets the menu's ID and InsertMenu inserts the menu into the menu list. (Setting the menu ID is for illustrative purposes only because the ID is not used in this demonstration. Since the system handles the standard Window menu automatically, an ID is ordinarily only required for menu adjustment purposes when the menu has been customised.)

The sixth block inserts a further single submenu (to be attached to the second item in the Special menu) into the submenu portion of the menu list. GetNewMBar does not read in the resource descriptions of submenus, so the first step is to read in the 'MENU' resource with GetMenu. InsertMenu inserts a menu reference for this menu into the menu list at the location specified in the second parameter to this call. Using the constant hierMenu (-1) as the second parameter causes the menu to be installed in the submenu portion of the menu list.

The last line causes a checkmark to be set against the Size menu item corresponding to the initialised value of the global variable gCurrentFontSizeItem.

DrawMenuBar draws the menu bar.

Note that, in Carbon, the contents of the Apple Menu Items folder are automatically added to the Apple menu.

### doEvents

doEvents switches according to the type of low-level or Operating System event received. Further processing is called for in the case of mouse-down or Command key equivalents, these being central to the matter of menu handling.

At the keyDown case, a check is made of the modifiers field of the event structure to establish whether the Command key was also pressed at the time. If so, menu enabling/disabling is attended to before the call to MenuEvent establishes whether the character code is associated with a currently enabled menu or submenu item in the menu list. If a match is found, MenuEvent returns a long integer containing the menu ID in the high word and the item number in the low word, otherwise it returns 0 in the high word. This long integer is then passed to the function doMenuChoice.

### doMouseDown

doMouseDown first establishes the window and window part in which the mouse-down event occurred, and switches accordingly. This demonstration program is specifically concerned with mouse-downs in the menu bar and the content region of the window.

If the event occurred in this program's menu bar, menu enabling/disabling is attended to before the call to MenuSelect. MenuSelect tracks the user's actions until the mouse button is released, at which time it returns a long integer. If the user actually chose a menu item, this long integer contains the menu ID in the high word and the item number in the low word, otherwise it contains 0 in the high word. This long integer is passed to the function doMenuChoice.

If the mouse-down event occurred in the content region of the window, and if the window to which the mouse-down refers is not the front window, SelectWindow is called to effect basic window activation/deactivation.

### doAdjustMenus

doAdjustMenus is called when a mouse-down occurs in the menu bar and when examination of a key-down event reveals that a menu item's keyboard equivalent has been pressed. No action is required in this simple program.

Later demonstration programs contain examples of menu adjustment functions.

### doMenuChoice

doMenuChoice takes the long integer returned by the MenuSelect and MenuEvent calls, extracts the high word (the menu ID) and the low word (the menu item number) and switches according to the menu ID.

At the first two lines, the menu ID and the menu item number are extracted from the long integer. The next two lines will cause an immediate return if the high word equals 0, (meaning that either the mouse button was released when the pointer was outside the menu box or MenuEvent found no menu list match for the key pressed in conjunction with the Command key).

If the menu ID represents either the Font menu or one of the Font menu's submenus, the menu handling function doFontMenu is called. Otherwise, the function switches on the menu ID so that the appropriate individual menu handling function is called. Note the handling of the submenu attached to the second item in the Special menu (case mSubMenu).

The Window menu is handled automatically by the system.

MenuEvent and MenuSelect leave the menu title highlighted if an item was actually chosen. Accordingly, the last line unhighlights the menu title when the action associated with the user's drop-down menu choice is complete.

### doAppleApplicationMenu

doAppleApplicationMenu takes the short integer representing the menu item. If this value represents the first item in the Mac OS 8/9 Apple menu or Mac OS X Application menu (the inserted "About..." item), text representing this item is drawn in the scrolling display.

If other items in the Mac OS 8/9 Apple menu are chosen, the system automatically opens the chosen object and passes control that object.

### doFileMenu

doFileMenu handles choices from the File menu when the program is run on Mac OS 8/9. In this demonstration, only the Quit item is enabled, all other items having been disabled in the File menu's 'MENU' resource. When this item is chosen, the global variable gDone is set to true, causing termination of the program.

### doEditMenu

doEditMenu switches according to the menu item number, drawing text representing the chosen item in the window.

### doFontMenu

doFontMenu first gets a reference to the Font menu object. This, together with the menu item number, is passed in a call to the function GetFontFamilyFromMenuSelection. This function returns a reference to the font family and a value representing the font style. (A font family reference represents a collection of fonts with the same design characteristics, e.g., Arial, Arial Bold, Arial Italic, and Arial Bold Italic. Font style values are, for example, 0 = plain, 1 = bold, 2 = italic, 3 = bold italic).

The font family reference and the font style value are passed in calls to TextFont and TextFace, which will cause subsequent text drawing to be conducted in the specified font and style. The call to GetFontName gets the font's name from the font family reference and the function drawItemString draws that name in the window.

The remaining code is mainly concerned with checkmarking the newly-chosen Font menu item and submenu item, and unchecking the previously chosen items.

If the menu ID represents the Font menu (meaning that a menu item without an attached submenu was chosen), the previously chosen item is unchecked, a global variable stores the item number of the newly-chosen item preparatory to the next call to doFontmenu, and the newly chosen item is checked. If a submenu item has previously been chosen, and thus checked, it is unchecked.

If, on the other hand, the menuID represents one of the Font menu's submenus:

- If a submenu item has previously been chosen, that item is unchecked. A reference to the submenu object is assigned to a global variable, the menu item number is stored in another global variable preparatory to the next call to doFontmenu, and the newly chosen submenu item is checked.

- The next two lines uncheck the previously checked Font menu item.

- The for loop walks the Font menu looking for a match between item names and the font name previously extracted from the font family reference. When a match is found, the loop exits, the loop variable containing the item number where the match was found. This is stored in a global variable preparatory to the next call to doFontMenu, and is also passed in the call to CheckMenuItem to check that item.

- The last block gets the style name from the menu object and draws that next to the font name in the window.

### doSizeMenu

doSizeMenu switches according to the menu item chosen in the Size menu, sets the text size for all text drawing to that size, unchecks the current size item, and checks the newly chosen item. gCurrentSize is then set to the chosen menu item number before the function returns.

### doSpecialMenu

doSpecialMenu handles a choice of the first item in the Special menu. Since the second item is the title of a submenu, only the first item is attended to in this function.

### doSubMenus

doSubMenus switches according to the chosen item in the submenu attached to the second menu item in the Special menu.

### drawItemString

The function drawItemString is incidental to the demonstration, being called by the menu selection handling functions to draw text in the application's window to reflect the user's menu choices.

## Demonstration Program Menus2 Listing

```
// *******************************************************************************************
// Menus2.c                                                                 CLASSIC EVENT MODEL
// *******************************************************************************************
//
// This program is based on Menus1.  The basic differences between this program and Menus1 are
// as follows:
//
// •  'xmnu' resources are used to extend the 'MENU' resources for some menus.
//
// •  Extended modifier keys (Shift, Option, and Control) are used to extend the Command-key
//    equivalents for two menu items in the Style menus.
//
// •  There are two Style menus (Style ('xmnu') and Style (Programmatic).  The two Style menus
//    are intended to demonstrate assigning extended modifier keys to a menu item (1) via an
//    'xmnu' resource and (2)  programmatically.
//
// •  Command IDs are assigned to all menu items except those in the system-managed menus and
//    the Font menu, and the associated menu handling code branches according to the command
//    ID of the chosen menu item (as opposed to menu ID and menu item).
//
// •  The Font menu is non-hierarchical.  It is also WYSIWYG, meaning that each item is drawn
//    in that font.
//
// •  The delete-to-the-left, delete-to-the-right, page-up, and page-down keys are assigned as
//    Command-key equivalents in the Size menu, and the glyphs are adjusted where necessary.
//
// •  The submenu is attached to the second item in the Special menu programmatically rather
//    than via the 'MENU' resource.
//
// •  Colour icons are included in the menu items in the submenu.
//
// •  Balloon help is provided, via 'hmnu' resources, for all menus.
//
// The extended modifier keys in the Style ('xmnu') menu are assigned via the 'xmnu' resource
// for that menu.  The extended  modifier keys in the Style (Programmatic) menu are assigned
// programmatically .
//
// The command IDs for items in the File, Edit, and Style ('xmnu') menus are assigned via the
// 'xmnu' resources for those menus.  The command IDs for the items in the Style
// (Programmatic), Size, and Special menus, and the submenu, are assigned programmatically.
//
// The colour icon in the first submenu item is assigned via the 'MENU' resource.  The colour
// icon in the second item is assigned programmatically via a call to
// SetMenuItemIconHandle.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  An 'MBAR' resource (preload, non-purgeable).
//
// •  'MENU' resources for the drop-down menus and submenu (all preload, all non-purgeable).
//
// •  'xmnu' resources (preload, purgeable) for the drop-down menus (except the system-managed
//     menus and the Font menu) and the submenu.
//
// •  'hmnu' resources (purgeable) providing balloon help for menus and menu items.
//
// •  Two 'cicn' resources (purgeable) for the items in the submenu.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *******************************************************************************************
// .............................................................................................. includes

#include <Carbon.h>

// ............................................................................................... defines

#define rMenubar          128
#define mAppleApplication 128
#define mFile             129
#define  iQuit            12
#define mFont             131
#define mStyleXmnu        132
#define mStyleProg        133
#define  iPlain           1
#define  iBold            3
#define  iItalic          4
#define  iOutline         6
#define  iUnderline       5
#define  iShadow          7
#define mSize             134
#define  iTen             1
#define  iTwelve          2
#define  iEighteen        3
#define  iTwentyFour      4
#define mSpecial          135
#define  iFirst           1
#define  iSecond          2
#define mSubmenu          136
#define  iBat             1
#define  iBowl            2
#define rWindowResource   128
#define rColourIcon       258

// ...................................................................................... global variables

Boolean       gRunningOnX          = false;
Boolean       gDone;
MenuItemIndex gCurrentFontMenuItem = 0;
Style         gCurrentStyle        = 0;
MenuItemIndex gCurrentSizeMenuItem = 2;

// ................................................................................... function prototypes

void  main                 (void);
void  doPreliminaries      (void);
OSErr  quitAppEventHandler (AppleEvent *,AppleEvent *,SInt32);
void  doGetMenus           (void);
void  doEvents             (EventRecord *);
void  doMouseDown          (EventRecord *);
void  doAdjustMenus        (void);
void  doMenuChoice         (SInt32);
void  doCommand            (MenuCommand);
void  doFontMenu           (MenuItemIndex);
void  doCheckStyleMenuItem (MenuID);
void  doCheckSizeMenuItem  (MenuItemIndex);
void  drawItemString       (Str255);

// ************************************************************************************* main

void  main(void)
{
  EventRecord eventStructure;
  WindowRef   windowRef;
  RGBColor    foreColour = { 0xFFFF,0xFFFF,0xFFFF };
  RGBColor    backColour = { 0x4444,0x4444,0x9999 };
  Rect        portRect;

  // .................................................................................. do preliminaries
  doPreliminaries();

  // ......................................................................................... open a window

  if(!(windowRef = GetNewCWindow(rWindowResource,NULL,(WindowRef) -1)))
  {
    SysBeep(10);
    ExitToShell();
  }

  SetPortWindowPort(windowRef);
  TextSize(10);
  RGBBackColor(&backColour);
  RGBForeColor(&foreColour);

  // .................................................. set up menu bar and menus, then show window

  doGetMenus();
  ShowWindow(windowRef);
  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);

  // ............................................................................................ event loop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,180,NULL))
      doEvents(&eventStructure);
  }
}

// ************************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(32);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                          NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                          0L,false);
  if(osError != noErr)
    ExitToShell();
}

// *************************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                            &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;
  return osError;
}

// ****************************************************************************** doGetMenus

void  doGetMenus(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  OSStatus      osError;
  ItemCount     hierMenuCount;
  SInt16        a, numberOfItems, fontNumber;
  Str255        fontName, smallSystemFontName;
  CIconHandle   cicnHdl;

  // ..................................................................... get and set menu bar

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }

  // ........................................................ set up Font menu and make WYSIWYG

  GetFontName(kThemeSmallSystemFont,smallSystemFontName);

  menuRef = GetMenuRef(mFont);
  if(menuRef != NULL)
  {
    osError = CreateStandardFontMenu(menuRef,0,0,kNilOptions,&hierMenuCount);
    if(osError == noErr)
    {
      numberOfItems = CountMenuItems(menuRef);
      for(a=1;a<=numberOfItems;a++)
      {
        GetMenuItemText(menuRef,a,fontName);
        GetFNum(fontName,&fontNumber);
        SetMenuItemFontID(menuRef,a,fontNumber);

        if(EqualString(fontName,smallSystemFontName,false,false))
        {
          CheckMenuItem(menuRef,a,true);
          gCurrentFontMenuItem = a;
        }
      }
    }
    else ExitToShell();
  }
  else
    ExitToShell();

  // ............... programmatically set the extended modifiers in Style (Programmatic) menu
```

```
menuRef = GetMenuRef(mStyleProg);
SetMenuItemModifiers(menuRef,iOutline,kMenuShiftModifier + kMenuOptionModifier
                     + kMenuControlModifier);
SetMenuItemModifiers(menuRef,iShadow,kMenuShiftModifier + kMenuOptionModifier);

// ………… insert submenu into menu list and programmatically attach it to Special menu, item 2

menuRef = GetMenu(mSubmenu);
if(menuRef != NULL)
{
  InsertMenu(menuRef,hierMenu);
  menuRef = GetMenuRef(mSpecial);
  SetMenuItemHierarchicalID(menuRef,iSecond,mSubmenu);
}
else
  ExitToShell();

// ……………… programmatically set command IDs for second Style, Size, Special menus and submenu

menuRef = GetMenuRef(mStyleProg);
SetMenuItemCommandID(menuRef,iPlain,     'plai');
SetMenuItemCommandID(menuRef,iBold,      'bold');
SetMenuItemCommandID(menuRef,iItalic,    'ital');
SetMenuItemCommandID(menuRef,iUnderline, 'unde');
SetMenuItemCommandID(menuRef,iOutline,   'outl');
SetMenuItemCommandID(menuRef,iShadow,    'shad');

menuRef = GetMenuRef(mSize);
SetMenuItemCommandID(menuRef,iTen,       'ten ');
SetMenuItemCommandID(menuRef,iTwelve,    'twel');
SetMenuItemCommandID(menuRef,iEighteen,  'eigh');
SetMenuItemCommandID(menuRef,iTwentyFour,'twen');

menuRef = GetMenuRef(mSpecial);
SetMenuItemCommandID(menuRef,iFirst,     'firs');

menuRef = GetMenuRef(mSubmenu);
SetMenuItemCommandID(menuRef,iBat,       'bat ');
SetMenuItemCommandID(menuRef,iBowl,      'bowl');

// ………………………………………………………………… programmatically set the icon for the Bowl item in the submenu

cicnHdl = GetCIcon(rColourIcon);
SetMenuItemIconHandle(menuRef,iBowl,kMenuColorIconType,(Handle) cicnHdl);

// ………………… programmatically set Command-key equivalents to Size menu items and adjust glyphs

menuRef = GetMenuRef(mSize);
SetItemCmd(menuRef,iTen,0x08);
SetMenuItemKeyGlyph(menuRef,iTen,kMenuDeleteLeftGlyph);
SetItemCmd(menuRef,iTwelve,0x7f);
SetMenuItemKeyGlyph(menuRef,iTwelve,kMenuDeleteRightGlyph);
SetItemCmd(menuRef,iEighteen,0x0b);
SetMenuItemKeyGlyph(menuRef,iEighteen,kMenuPageUpGlyph);
SetItemCmd(menuRef,iTwentyFour,0x0c);
SetMenuItemKeyGlyph(menuRef,iTwentyFour,kMenuPageDownGlyph);

// …………………………… programmatically exclude the mark column and set the font in the Special menu

menuRef = GetMenuRef(mSpecial);
SetMenuExcludesMarkColumn(menuRef,true);

GetFNum("\pGadget",&fontNumber);
if(fontNumber != 0)
  SetMenuFont(menuRef,fontNumber,12);

// …………………………………………………………………… if running on Mac OS X, create Help menu and insert one item

if(gRunningOnX)
```

```
  {
    HMGetHelpMenu(&menuRef,NULL);
    InsertMenuItem(menuRef,"\pMenus Help",0);
    SetMenuItemCommandID(menuRef,1,'help');
  }

  // .................................................. set initial font, style, and size, and checkmark them

  doCheckStyleMenuItem(mStyleXmnu);
  doCheckStyleMenuItem(mStyleProg);
  doCheckSizeMenuItem(iTen);

  // ......................................................................................... draw menu bar

  DrawMenuBar();
}

// ********************************************************************************* doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      break;

    case updateEvt:
      BeginUpdate((WindowRef) eventStrucPtr->message);
      EndUpdate((WindowRef) eventStrucPtr->message);
      break;
  }
}

// ****************************************************************************** doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;
  SInt32         menuChoice;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doAdjustMenus();
      menuChoice = MenuSelect(eventStrucPtr->where);
      doMenuChoice(menuChoice);
      break;

    case inContent:
      if(windowRef != FrontWindow())
        SelectWindow(windowRef);
      break;
    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where))
        gDone = true;
      break;
  }
}

// ***************************************************************************** doAdjustMenus

void  doAdjustMenus(void)
{
  // Adjust menus here.  Use EnableMenuCommand and DisableMenuCommand to enable/disable those
  // menu items with command IDs.
}

// ****************************************************************************** doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  OSErr         osErr;
  MenuCommand   commandID;

  menuID   = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;
  else if(menuID == mFont)
    doFontMenu(menuItem);
  else
  {
    osErr = GetMenuItemCommandID(GetMenuRef(menuID),menuItem,&commandID);
    if(osErr == noErr && commandID != 0)
      doCommand(commandID);
  }

  HiliteMenu(0);
}

// ********************************************************************************* doCommand

void  doCommand(MenuCommand commandID)
{
  MenuRef menuRef;

  switch(commandID)
  {
    // .............................................................. Apple/Application menu

    case 'abou':                                                                // About
      drawItemString("\pAbout Menus2");
      break;

    // ............................................................................ File menu

    case 'quit':                                                                 // Quit
      gDone = true;
      break;

    // ............................................................................ Edit menu

    case 'undo':                                                                 // Undo
      drawItemString("\pUndo");
      break;

    case 'cut ':                                                      // Cut
      drawItemString("\pCut");
      break;

    case 'copy':                                                     // Copy
      drawItemString("\pCopy");
      break;

    case 'past':                                                    // Paste
      drawItemString("\pPaste");
      break;

    case 'clea':                                                    // Clear
      drawItemString("\pClear");
      break;

    // .......................................... Style ('xmnu') and Style (Programmatic) menu

    case 'plai':                                                    // Plain
      gCurrentStyle = 0;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'bold':                                                     // Bold
      if(gCurrentStyle & bold)
        gCurrentStyle -= bold;
      else
        gCurrentStyle |= bold;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'ital':                                                  // Italics
      if(gCurrentStyle & italic)
        gCurrentStyle -= italic;
      else
      gCurrentStyle |= italic;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'unde':                                                // Underline
      if(gCurrentStyle & underline)
        gCurrentStyle -= underline;
      else
        gCurrentStyle |= underline;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'outl':                                                  // Outline
      if(gCurrentStyle & outline)
        gCurrentStyle -= outline;
      else
        gCurrentStyle |= outline;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'shad':                                                   // Shadow
      if(gCurrentStyle & shadow)
        gCurrentStyle -= shadow;
      else
        gCurrentStyle |= shadow;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
```

```
      break;

    // ............................................................................ Size menu

    case 'ten ':                                                                     // 10
      TextSize(10);
      doCheckSizeMenuItem(iTen);
      break;

    case 'twel':                                                                     // 12
      TextSize(12);
      doCheckSizeMenuItem(iTwelve);
      break;

    case 'eigh':                                                                     // 18
      TextSize(18);
      doCheckSizeMenuItem(iEighteen);
      break;

    case 'twen':                                                                     // 24
      TextSize(24);
      doCheckSizeMenuItem(iTwentyFour);
      break;

    // ......................................................................... Special menu

    case 'firs':                                                                  // First
      drawItemString("\pFirst Item");
      break;

    // .............................................................................. submenu

    case 'bat ':                                                                    // Bat
      menuRef = GetMenuRef(mSubmenu);
      DisableMenuItem(menuRef,iBat);
      EnableMenuItem(menuRef,iBowl);
      drawItemString("\pBat");
      break;

    case 'bowl':                                                                   // Bowl
      menuRef = GetMenuRef(mSubmenu);
      DisableMenuItem(menuRef,iBowl);
      EnableMenuItem(menuRef,iBat);
      drawItemString("\pBowl");
      break;

    case 'help':
      AHGotoPage(CFSTR("Menus Help"),CFSTR("Menus.htm"),NULL);
      break;
  }
}

// ********************************************************************************** doFontMenu

void  doFontMenu(MenuItemIndex menuItem)
{
  MenuRef      menuRef;
  OSStatus     osError;
  FMFontFamily currentFontFamilyReference;
  FMFontStyle  fontStyle;
  Str255       fontName;

  menuRef = GetMenuRef(mFont);

  osError = GetFontFamilyFromMenuSelection(menuRef,menuItem,&currentFontFamilyReference,
            &fontStyle);

  if(osError == noErr || osError == menuPropertyNotFoundErr)
  {
    TextFont(currentFontFamilyReference);

    CheckMenuItem(menuRef,gCurrentFontMenuItem,false);
    gCurrentFontMenuItem = menuItem;
    CheckMenuItem(menuRef,gCurrentFontMenuItem,true);

    GetMenuItemText(menuRef,menuItem,fontName);
    drawItemString(fontName);
  }
  else
    ExitToShell();
}

// ********************************************************************** doCheckStyleMenuItem

void  doCheckStyleMenuItem(MenuID menuID)
{
  MenuRef        styleMenuRef;
  static Boolean stringAlreadyDrawnOnce = false;

  styleMenuRef = GetMenuRef(menuID);

  CheckMenuItem(styleMenuRef,iPlain,    gCurrentStyle == 0);
  CheckMenuItem(styleMenuRef,iBold,      gCurrentStyle & bold);
  CheckMenuItem(styleMenuRef,iItalic,    gCurrentStyle & italic);
  CheckMenuItem(styleMenuRef,iUnderline,gCurrentStyle & underline);
  CheckMenuItem(styleMenuRef,iOutline,  gCurrentStyle & outline);
  CheckMenuItem(styleMenuRef,iShadow,    gCurrentStyle & shadow);

  TextFace(gCurrentStyle);

  if(!stringAlreadyDrawnOnce)
    drawItemString("\pStyle change");

  stringAlreadyDrawnOnce = !stringAlreadyDrawnOnce;
}

// *********************************************************************** doCheckSizeMenuItem

void  doCheckSizeMenuItem(MenuItemIndex menuItem)
{
  MenuRef sizeMenuRef;

  sizeMenuRef = GetMenuRef(mSize);

  CheckMenuItem(sizeMenuRef,gCurrentSizeMenuItem,false);
  CheckMenuItem(sizeMenuRef,menuItem,true);

  gCurrentSizeMenuItem = menuItem;

  drawItemString("\pSize change");
}

// **************************************************************************** drawItemString

void  drawItemString(Str255 eventString)
{
  RgnHandle tempRegion;
  WindowRef windowRef;
  Rect      scrollBox;

  windowRef = FrontWindow();
  tempRegion = NewRgn();

  GetWindowPortBounds(windowRef,&scrollBox);

  ScrollRect(&scrollBox,0,-30,tempRegion);
  DisposeRgn(tempRegion);
  MoveTo(8,286);
  DrawString(eventString);
}

// ******************************************************************************************
```

## Demonstration Program Menus2 Comments

When this program is run, the user should choose Show Balloons from the Help menu and make menu choices from all menus, including the Apple menu. Choices should be made using the mouse and, where appropriate, the keyboard equivalents. The user should note:

- The extended modifier keys assigned to the last two items in the Style menus.
- The Command-key equivalents assigned to the items in the Size menu. (These are, in order, delete-to-the-left key, delete-to-the-right key, page-up key, and page-down key.)
- That the Font menu is WYSIWYG.
- That the marking character column has been deleted from the Special menu and the menu items in this menu are drawn in the Gadget font (assuming it is available).
- That the items in the submenu attached to the second item in the Special menu have colour icons.
- The balloon help provided for all menus and menu items.

The Menus2 demonstration program package also includes a demonstration of Apple Help, including the methodology used to create an item in the Mac OS 8/9 Help menu. The Apple Guide file titled "Menus Guide", which will cause a "Menus Help" item to be created in the Mac OS 8/9 Help menu, should be retained in the same folder as the Menus2 application. An alias of the folder titled "Menus Help" should be placed in the Help folder in the System Folder (Mac OS 8/9) and in the user's Help folder (~/Library/Documentation/Help) (Mac OS X). You will then be able to access the help content by choosing Menus Help from the Help menu.

The help content does not provide user assistance for Menus2 programs as such. Rather, it provides a brief description of how to provide user assistance for your application using Apple Help.

Because this demonstration program is based on Menus1, the following comments exclude those for the functions that remain unchanged.

### main

The calls to RGBBackColor and RGBForeColor set the window background and foreground colours to, respectively, dark blue and white.

### doGetMenus

doGetMenus sets up the menu bar and the various menus.

GetNewMBar reads in the 'MENU' resources for each menu specified in the 'MBAR' resource and creates a menu object for each of those menus. (Note that the error handling here and in other areas of this program is somewhat rudimentary: the program simply terminates.) SetMenuBar makes the newly created menu list the current list.

The next block utilizes the Menu Manager function CreateStandardFontMenu in the creation of a non-hierarchical Font menu. Following the call to CreateStandardFontMenu, the process of making the menu WYSIWYG begins. The call to CountMenuItems returns the number of items in the menu. Then, for each of these items, GetMenuItemText gets the font's name, GetFNum gets the font number associated with the font name, and SetMenuItemFontID sets the font for the menu item. In the following if block, the current item is checkmarked if the item name equals the name of the small system font, and the global variable which keeps track of the currently selected font is assigned the item number.

The next block programmatically assigns extended modifier keys to the Outline and Shadow items in the Style (Programmatic) menu. The SetMenuItemModifiers calls assign Shift-Option-Control to the Outline item and Shift-Option to the Shadow item. (The extended modifier keys for the same two items in the Style ('xmnu') menu are assigned in the associated 'xmnu' resources.)

The next block inserts the application's single submenu into the submenu portion of the menu list and programmatically attaches it to the Special menu's second menu item. GetNewMBar does not read in the resource descriptions of submenus, so the first step is to read in the 'MENU' resource with GetMenu. InsertMenu inserts a menu object for this menu into the menu list at the location specified in the second parameter to this call. (Using the constant hierMenu (-1) as the second parameter causes the menu to be installed in the submenu portion of the menu list.) The call to GetMenuRef gets a reference to the

Special menu, which is used in the following call to SetMenuHierarchicalID to attach the submenu to the second item in the Special menu.

The following rather large block programmatically assigns command IDs to all items in the Style (Programmatic), Size, and Special menus and the submenu. (Command IDs for the File and Style ('xmnu') menus are assigned in the associated 'xmnu' resources. It is not possible to assign command IDs to the items in the Font menu.) The Command IDs are defined in the four-character-code format, which packs four one-byte characters together in a 32-bit value. For example, 'plai' expressed as hexadecimal is 0x706C6169. 70 is the ASCII code for p, 6C is the ASCII code for l, and 69 is the ASCII code for i.

The following block programmatically assigns a colour icon to the second item in the submenu. The call to GetCIcon creates a CIcon data structure and initializes it from data read in from the specified 'cicn' resource. The handle to this structure is then passed as the last parameter in the SetMenuItemIconHandle, the third parameter specifying that the type of icon is a colour icon. (The colour icon for the first item in the submenu is assigned in the associated 'xmnu' resource.)

The next block programmatically assigns command-key equivalents to the items of the Size menu. (Because the keys assigned are the two delete keys and the page-up and page-down keys, it is not possible to make these assignments within the 'MENU' resource.) Also, a substitute glyph must be assigned, otherwise the correct glyphs will not be displayed. The calls to SetItemCmd assign the specified key to the menu item, and a substitute glyph is assigned via calls to SetMenuItemGlyph. If this is not done, the glyphs displayed will not be the correct visual representations of the keys. (These substitute glyphs could also have been specified in the keyboard glyph fields for these items in the menu's 'xmnu' resource.)

In the next block, SetMenuExcludesMarkColumn is called to delete the marking character column from the Special menu and SetMenuFont is called to set the font for the menu items in this menu to Gadget (assuming that font is present).

In the next block, and only if the program is running on Mac OS X, HMGetHelpMenu is called to create a Help menu, InsertMenuItem is called to insert a single item in that menu, and SetMenuItemCommandID assigns a command ID to that item.

The next block sets checkmarks against the appropriate font, style and size menu items according to the initialised values of the associated global variables.

The call to DrawMenuBar draws the menu bar

Note that, in Carbon, the contents of the Apple Menu Items folder are automatically added to the Apple menu.

### doMenuChoice

doMenuChoice extracts the menu ID and menu item number from the long integer returned by the MenuSelect and MenuEvent calls. An immediate return is made if the high word equals 0. The function "special cases" the Font menus, calling the function for handling choices from that menu. Otherwise, GetMenuItemCommandID is called. GetMenuItemCommandID returns zero as the function result if the call is successful, and a pointer to an integer representing the value of the item's command ID will be returned in the third parameter. If the call is successful, and if a zero is not returned in the third parameter, a command ID exists for the item. Accordingly, the command ID is passed in a call to the function doCommand.

MenuSelect and MenuEvent leave the menu title highlighted if an item was actually chosen. Accordingly, the last line unhighlights the menu title when the action associated with the user's drop-down menu choice is complete.

### doCommand

doCommand handles choices from those menus whose items have command IDs.

Note that the initial handling of all of the remaining menu items, regardless of which menu they belong to, is attended to within the one switch in the one function. The responses to the user choosing the various menu items is the same as in Menus1, except that the code relating to checkmarking the Style menu items has been added and the code for checkmarking the Size menu items and storing the current size has been divided between this function a further handling function (doCheckSizeMenuItem).

At the block titled Style ('xmnu') and Style (Programmatic) menu, bits in the global variable gCurrentStyle are set or unset according to the font styles selected. The code reflects the fact that Bold, Italic, Underline, Outline and Shadow style selections are additive, not mutually exclusive, and that a selection of Plain must unset all bits in gCurrentStyle. The code also reflects the requirement that, except in the case of the Plain item, the selection of a checked item must cause that item to be unchecked, and vice versa. With gCurrentStyle set, the function doCheckStyleMenuItem is called to check/uncheck the relevant menu items as appropriate.

Note that the handling of the two submenu items has been changed to make the items mutually exclusive.

The 'help' command ID case applies only when the program is run on Mac OS X. The function AHGoToPage is called to deliver a request to load the specified HTML file in the specified Help book folder to the Help Viewer application.

### *doCheckStyleMenuItem*

doCheckStyleMenuItem is called from doMenuChoice when an item in the Style menu is chosen. With the appropriate bit settings of gCurrentStyle attended to within doMenuChoice, a reference to the Style menu object is obtained. This is required for the six CheckMenuItem calls, which check or uncheck the individual menu items according to whether the third parameter evaluates to, respectively, true or false.

The call to TextFace sets the style for subsequent text drawing. The last line draws some text to prove that the desired effect was achieved.