3

MENUS

Includes Demonstration Program Menus1 and Menus2

Introduction

Types of Menus

A menu is a user interface element which 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 is identified by a menu title in the menu bar. Each pull-down menu comprises a menu title 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 which does not appear in the menu bar but rather appears on another part of the screen when the user presses the mouse button while the cursor is at a particular location.

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, insert scrolling indicators as items in a menu, calculate the menu's dimensions, etc. An MDEF thus determines the look and behaviour of menus.

- ***Menu Bar Definition Function.*** The Menu Manager uses the menu bar definition function to draw and clear the menu bar, determine whether the cursor is currently within the menu bar or any currently displayed menu, calculate the left edges of menu titles, highlight a menu title, invert the entire menu bar, draw the menu's shadow box, and save/restore the bits behind a menu. A menu bar definition function thus determines the look and behaviour of the menu bar.

Standard Appearance-Compliant Menu and Menu Bar Definition Functions

The system software provides a standard Appearance-compliant MDEF and a standard Appearance-compliant menu bar definition function, which are stored as code resources in the System file. The standard MDEF is the 'MDEF' resource with a resource ID of 63. The standard menu bar definition function is the 'MBDF' resource with a resource ID of 63.

> ***Historical Note***
>
> These are the resource IDs of revised definition functions first issued with Mac OS 8 and the Appearance Manager. The old pre-Appearance definition functions have resource IDs of 0, and remain in the System file. In Mac OS 8.0 and 8.1, the new definition functions are located in the Appearance extension. As of Mac OS 8.5, they are located in the System file.

When you define your menus and menu bar, you specify the definition functions that the Menu Manager should use when managing them. Ordinarily, you will use the standard 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.[1] Given the new features added by the latest standard definition functions, however, it is all but impossible to conceive of circumstances that would require the writing of a custom definition function.

The Menu Bar and Menus

The Menu Bar

The menu bar extends across the top of the screen. As defined by the standard menu bar definition function, the menu bar is white and high enough to display menu titles in the height of the large system font[2] plus a single pixel bottom border.

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 for a menu command) 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 menu's 'MBAR' resource should be defined such that the Apple menu is the first menu in the menu bar, with the File and Edit menus being the next two. The Help and Application menus 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 provide, as a minimum, the **standard menus**. The standard menus are the Apple menu, the File menu and the Edit menu.

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.

---

1 Chapter 20 — Lists and Custom List Definition Functions includes an example of custom definition functions for another element of the user interface known as a list.

2 For Roman scripts, Chicago or Charcoal (Mac OS 8.0, 8.1, and 8.5) plus Capitals, Gadget, Sand, Techno, or Textile (Mac OS 8.5))

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 was introduced with Mac OS 8 and the Appearance Manager and is, in effect, the 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

> ***Historical Note***
>
> ***The 'mctb' Resource.*** The use of the old standard MDEF (resource ID 0) was sometimes accompanied by the use of a menu colour table ('mctb') resource with the same ID as the associated 'MENU' resource. This allowed the default colours of the title, item text, and background to be changed. The use of 'mctb' resources is inconsistent with the concept of themes and Appearance-compliance; accordingly, under the Appearance Manager, the use of this resource is not recommended.

Menu Items

A menu item can contain text or a dividing line (that is, a **divider**). A divider is always dimmed. Each menu item, other than dividers, can have a number of characteristics as follows:

- An icon, small icon, reduced icon, colour icon, or an icon from an icon family[3] to the left of the menu item's text. (Note that items with small or reduced icons cannot have submenus.)

> ***Historical Note***
>
> Support for icon families was introduced with Mac OS 8 and the Appearance Manager.

- A checkmark or other **marking character** to the left of the menu item's text, indicating the status of the menu item or the mode it controls. (A menu item can have a mark or a submenu, but not both.)

- The 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.)

- 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 marking character, 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 box requesting more information from the user. (The ellipsis character should not be used in menu items which 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.)

3 The various icon types are described at Chapter 13— Offsreeen Graphics Worlds, Pictures, Cursors, and Icons.

A menu can contain any number of menu items; however, in versions of the system software earlier than Mac OS 8.5, only the first 31 can be disabled.

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 which perform actions and commands which set attributes. The action commands which are logically related should be grouped, as should attribute commands which are interdependent. The attribute commands which are mutually exclusive, and those which 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**[4]. You 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 so, 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.[5]

> ***Historical Note***
>
> Prior to Mac OS 8 and the Appearance Manager, only **Command-key equivalents** for menu items were supported. Support for keyboard equivalents was introduced with Mac OS 8 and the Appearance Manager. This new capability is generally referred to as "support for extended modifier keys" (Shift, Option, and Control).

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-O	Open…	File
Command-P	Print…	File
Command-Q	Quit	File
Command-S	Save	File
Command-V	Paste	Edit
Command-W	Close	File
Command-X	Cut	Edit
Command-Z	Undo	Edit

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

4 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.

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

Menus Added Automatically By the Menu Manager

The menus added automatically by the Menu Manager (the Help and Application menus) are sometimes referred to as the **system-managed menus**. The Application menu has an icon as a title. The Help menu is displayed only if space is available; however, the application menu is invariably displayed, overlapping the main part of a long menu if this becomes necessary.

> ***Historical Note***
>
> Prior to Mac OS 8 and the Appearance Manager, the Help menu had an icon as a title and was located immediately to the left of the Application menu. The new menu bar definition function uses Help as the menu title and displays the menu immediately to the right of your application's last menu.

Your application does not need to take any action if the user chooses an item from the Application menu. However, if the user chooses an item added by your application to the Help menu, your application is responsible for taking the appropriate action.

The Apple Menu

The Apple menu should be defined as the first in your application. Typically, applications provide an About command as the first menu item, followed by a divider. The remaining items are, of course, controlled by the contents of the Apple Menu Items folder in the System folder.

To create your application's Apple menu, firstly define the Apple menu title, the characteristics of your application's About command and the divider following it in a 'MENU' resource. Then insert the contents of the Apple Menu Items folder into your application's Apple menu by calling AppendResMenu, with 'DRVR' specified as the resource type in the parameter theType.

When the user chooses the About command, your application should display a dialog box or an alert box containing your application's name, version number, copyright information, any other information as required, and an OK button.

When the user chooses an item other than the About command, your application should call the OpenDeskAcc function, which schedules the chosen item for execution and then returns to your application. At the next call to WaitNextEvent, your application receives a suspend event and the chosen item becomes the foreground process.

The File Menu

The standard File menu contains commands related to the management of documents, plus the Quit command. The standard commands (see Fig 1) should be supported by your application. Any other commands added to the standard section of the menu should pertain to the management of documents.

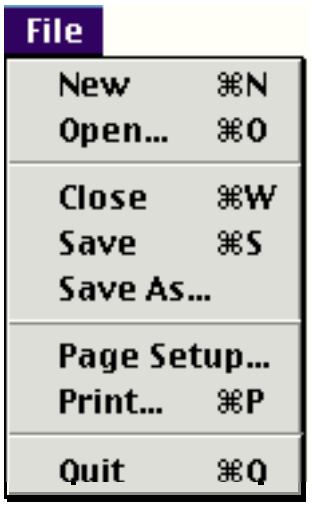

FIG 1 - STANDARD FILE

The actions your application should take when File menu commands are chosen are detailed at Chapter 15 — Printing and Chapter 16 — Files.

The Edit Menu

The standard Edit menu (see Fig 2) provides commands which allow the user to edit the contents of their documents, to copy data between different applications using the Clipboard, and to facilitate data sharing between documents created by different applications via publish and subscribe.

FIG 2 - STANDARD EDIT I

All Macintosh applications should include the standard editing commands (Undo, Cut, Copy, Paste and Clear) so as to support those operations in dialog boxes.

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 Help Menu

You can add items to the end of the Help menu to give the user access to any online help that your application provides in addition to help balloons. Items are added to the Help menu using HMGetHelpMenuHandle and AppendMenu. When adding items, include the name of your application in the command so as to indicate to the user just which application the help relates to.

Help Balloons

In the Help menu, the effect of selecting Show Balloons and Hide Balloons is global and affects all applications. The Help Manager provides balloons for the Apple, Help and Application menu titles, for items in the Application menu, and for the standard items in

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

The Application Menu

When the user chooses an item from the Application menu, the Menu Manager handles the event as appropriate. For example, if the user chooses another application, the Menu Manager sends your application a suspend event.

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 are added to the Font menu using AppendResMenu or InsertResMenu, which add items to the specified menu in alphabetical order.

Your application should indicate which font is in use by adding a checkmark to the left of the 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.

Font Attributes

Separate menus should be used to accommodate lists of font attributes such as styles and sizes. Since the system software supports both bitmapped and TrueType fonts, your application should not provide an upper limit for font sizes.

WYSIWYG Font Menus

Mac OS 8 and the Appearance Manager introduced the new function SetMenuItemFontID, which enables 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. Hierarchical menus should be used to provide the user with additional choices in the nature of attributes. They should not be used to provide additional commands. There should only ever be one hierarchical level, that is, there should be only one level of submenus.

Pop-Up Menus

Pop-up menus are used to present the user with a list of choices in a dialog box or window. 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 3) and behaviour is thus determined by a pop-up menu button control definition function.

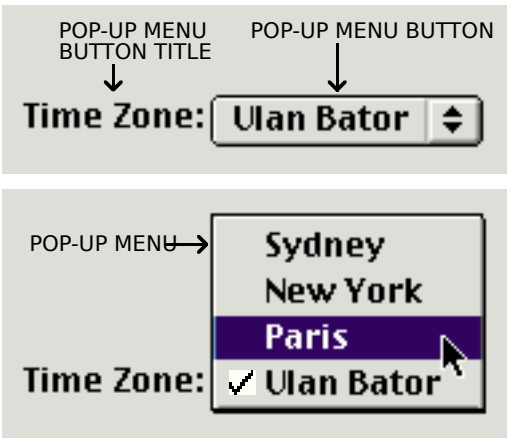

FIG 3 - POP-UP MENU BUTTON (EXAMI

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

Menu Structures, Menu IDs and Item Numbers, Command IDs, and Menu Lists

The Menu Structure

The Menu Manager maintains information about menus in **menu structures**, a data structure of type MenuInfo:

```
TYPE
    MenuInfoPtr = ^MenuInfo;
    MenuInfo = RECORD
                menuID:             INTEGER;
                menuWidth:          INTEGER;
                menuHeight:         INTEGER;
                menuProc:           Handle;
                enableFlags:        LONGINT;
                menuData:           Str255;
END;

MenuPtr = ^MenuInfo;
MenuHandle = ^MenuPtr;
```

You typically specify most of this information in a 'MENU' resource. When you create a menu, the Menu Manager creates a menu structure for the menu and returns a handle to that structure. The Menu Manager automatically updates the menu structure 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 handle to the menu's menu structure. 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 must use a menu ID of from 1 to 235.

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

Command IDs

Mac OS 8 and the Appearance Manager introduced the **command ID**, which 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 handles to the menu structures of one or more menus (although a menu list can, in fact, be empty). The end of a menu list contains handles to the menu structures of submenus and pop-up menus, if any, the phrase "submenu portion of the menu list" referring to this portion of the list.

When your application initialises the Menu Manager, 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 — System Software, Memory, And Resources, 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. Macintosh Pascal 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 4 shows the structure of a compiled 'MBAR' resource. The number of resource IDs should match the number of menus declared in the first two bytes.

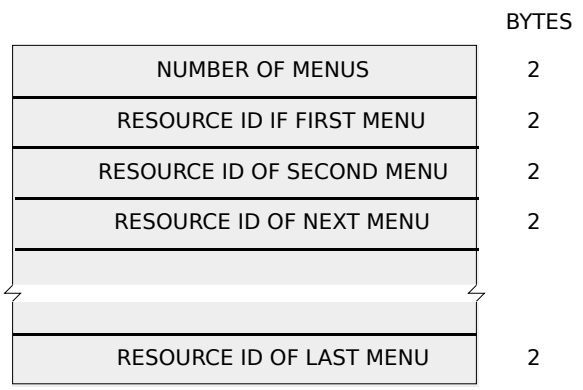

FIG 4 - STRUCTURE OF A COMPILED 'MBAR'

Structure of a Compiled 'MENU' Resource

Fig 5 shows the structure of a compiled 'MENU' resource (and its variable length data) and how it "feeds" the menu structure.

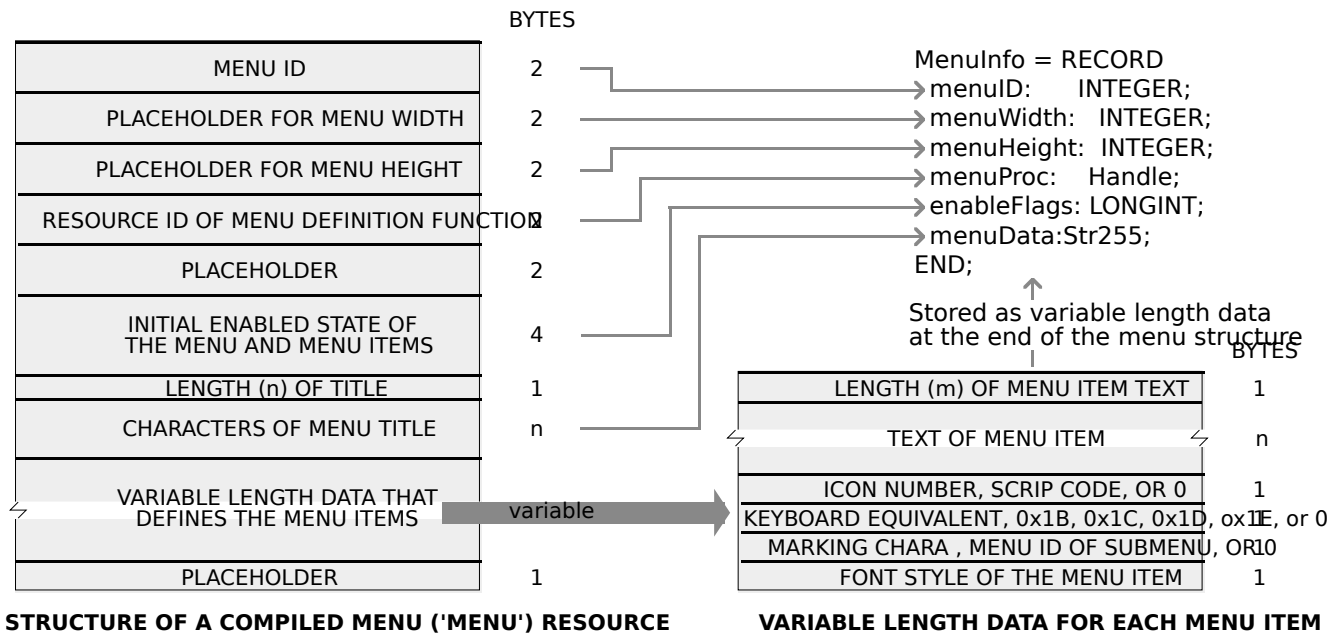

FIG 5 - STRUCTURE OF A COMPILED MENU ('MENU') RESOURCE AND ITS VARIABLE LENG

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

Field	Description
MENU ID	Each menu in your application should have a unique menu ID. Note that the menu ID does not have to match the resource ID, although by convention most applications assign the same number for a menu's resource ID and menu ID. A menu ID from 1 to 235 indicates a menu (or submenu) of an application. A negative menu ID indicates a menu belonging to a driver. A menu ID from 236 to 255 indicates a submenu of a driver. Apple reserves the menu ID of 0.
PLACEHOLDER FOR MENU WIDTH 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 menuWidth and menuHeight fields of the menu structure.
RESOURCE ID OF MENU DEFINITION FUNCTION	If the integer 63 appears here, the Menu Manager uses the standard MDEF to manage the menu. If you provide your own MDEF, its resource ID should appear in this field. After reading in the menu's resource data, the Menu Manager reads in the MDEF, if necessary. The Menu Manager stores a handle to the MDEF in the menuProc field of the menu structure.
INITIAL ENABLED STATE OF THE MENU AND MENU ITEMS	This is a 32-bit value where bits 1-31 indicate if the corresponding menu item is enabled or disabled, and bit 0 indicates whether the menu as a whole is enabled or disabled. The Menu Manager automatically enables menu items greater than 31 when a menu is created.
VARIABLE LENGTH DATA THAT DEFINES THE MENU ITEMS	The Menu Manager simply reads in this variable length data for each menu item and stores it as variable data at the end of the menu structure. The MDEF is responsible for interpreting the contents of the data (see Fig 5).

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 The icon number is a number from 1 to 255 (or from 1 to 254 for small or reduced icons). The Menu Manager adds 256 to the icon number to generate the resource ID of the menu item's icon. If the menu item specifies an

	icon, you should also provide a 'cicn' (colour icon) or 'ICON' resource with a resource ID equal to the icon number plus 256. If you want the Menu Manager to reduce an 'ICON' resource to the size of a small icon ('SICN'), or reduce the size of a 'cicn' resource by half, also provide the value 0x1D in the keyboard equivalent field (see below). If you want a 'SICN' resource, also provide the value 0x1E in the keyboard equivalent field. The Menu Manager looks first for a 'cicn' resource with the calculated resource ID. In the era of the Appearance Manager, colour icons are much to be preferred. SCRIPTCODE (Not to be used when the 'MENU' resource is extended with an 'xmnu' resource) If you want the Menu Manager to draw the item's text in a script other than the system script, specify the script code here and also provide 0x1C in the keyboard equivalent field (see below). *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.* 0 Indicates that the menu item does not contain an icon and uses the system script.
KEYBOARD EQUIVALENT, $1B, $1C, $1D, $1E, OR 0	KEYBOARD EQUIVALENT Specified as a one-byte character and, actually, a Command-key equivalent only. 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). $1B Indicates that the menu item has a submenu. (The menu ID of the submenu should be assigned to the marking character field (see below).) $1C (Not to be used when the 'MENU' resource is extended with an 'xmnu' resource) Indicates that the item uses a script other than the system script. (The script code should be assigned to the icon number field (see above).) *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.* $1D For menu items containing icons, causes the Menu Manager to reduce an 'ICON' resource to the size of a small icon, or reduce the size of a 'cicn' resource by half. $1E Indicates 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).) 0 Indicates 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 Special characters, such as the checkmark and diamond characters are available to indicate the marks associated with a menu item. MENU ID OF SUBMENU Submenus of an application must have menu IDs from 1 to 235. Submenus of a driver must have menu IDs from 236 to 255.. 0 Indicates that the item has neither a mark nor a submenu.
FONT STYLE OF THE MENU ITEM	Indicates whether the font style of the menu item should be plain, or any combination of bold, italic, outline, and shadow.

Structure of a Compiled 'xmnu' Resource

The 'xmnu' resource provides for the additional features introduced with Mac OS 8 and the Appearance Manager, for example, support for extended modifier keys, command IDs,

etc. Fig 6 shows the structure of a compiled 'xmnu' resource and an individual menu item entry in that resource.

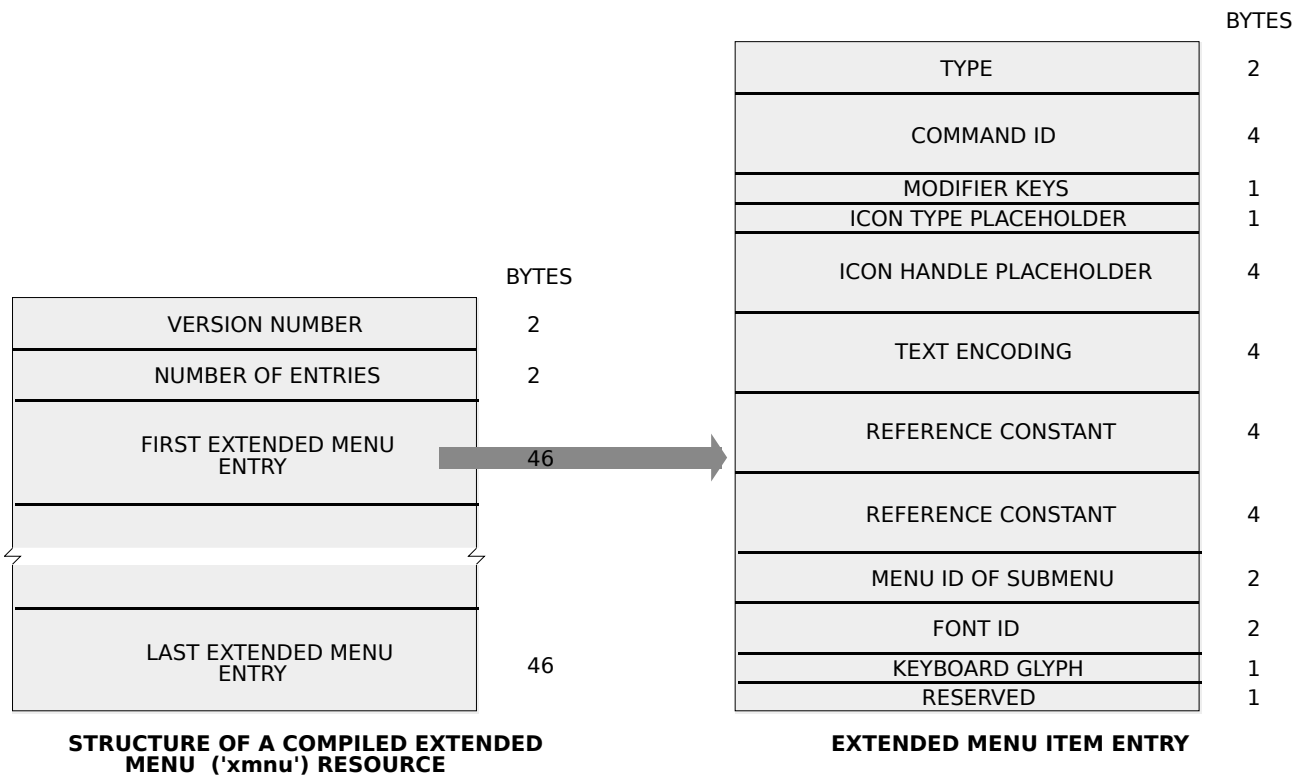

FIG 6 - STRUCTURE OF A COMPILED EXTENDED MENU ('xmnu') RESOURCE AND AN EXTEND

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

Field	Description
VERSION NUMBER	An integer specifying the version of the resource.
NUMBER OF ENTRIES	An integer that specifies the number of entries in the resource. Each entry is an extended menu item structure.
FIRST EXTENDED MENU ENTRY ... LAST EXTENDED MENU ENTRY	A series of extended menu item structures, each of which consists of a type, command ID, modifier keys, text encoding, reference constants, menu ID of submenu, font ID, and keyboard glyph. (A glyph is the visual representation of a character (see Chapter 19 — Text and TextEdit).

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 Menu Manager to read the rest of the structure. 0 indicates that there is no information for the item, causing the Menu Manager to skip the rest of the structure.
COMMAND ID	A unique value which you set to identify the menu item (instead of referring to it using the menu ID and item number). This value may be ascertained via a call to GetMenuItemCommandID. A command ID may be assigned to a menu item programmatically via a call to SetMenuItemCommandID.
MODIFIER KEYS	A mask that determines which modifier keys are used in a keyboard equivalent to select a menu item. The current modifier keys may be ascertained via a call to GetMenuItemModifiers. Modifier keys may be assigned to a menu item programmatically via a call to SetMenuItemModifiers.
ICON TYPE PLACEHOLDER ICON HANDLE PLACEHOLDER	Initially 0. Calls to SetMenuItemHandle assign the icon type and icon handle to these two fields.

TEXT ENCODING	A long integer which indicates the text encoding that the menu item text will use. This field of the 'xmnu' resource should be used instead of setting the keyboard equivalent field in the 'MENU' resource to $1C and the icon number field to the script code. If you want the text of the item to use the system script, this value should be -1. (This should be used as the default.) If you want the text of the menu item to use the current script, this value should be -2.. The current text encoding may be ascertained via a call to GetMenuItemTextEncoding. Text encoding may be assigned to a menu item programmatically via a call to SetMenuItemTextEncoding.
REFERENCE CONSTANT	Any value that an application wishes to store. The current value may be ascertained via a call to GetMenuItemRefCon. Reference constants may be assigned to a menu item programmatically via a call to SetMenuItemRefCon.
REFERENCE CONSTANT	Any additional value that an application wishes to store. The current value may be ascertained via a call to GetMenuItemRefCon. Reference constants may be assigned to a menu item programmatically via a call to SetMenuItemRefCon.
MENU ID OF SUBMENU	A value between 1 and 235, identifying the submenu. The current submenu ID may be acertained via a call to GetMenuItemHierarchicalID. 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	An integer representing the ID of the font family. If this value is 0, then the system font ID is used. The current font ID may be acertained via a call to GetMenuItemFontID. 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. The current keyboard glyph may be ascertained via a call to GetMenuItemGlyph. If the value in this field is zero, the keyboard glyph uses the keyboard font. Use of the keyboard font (rather than the system font) provides a consistent user interface across applications, since a modifier key's symbol will not change regardless of what system font is running. If the value in this field is nonzero, you can override the character code to be displayed with a substitute glyph. 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 introduced with Mac OS 8 and the Appearance Manager.

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

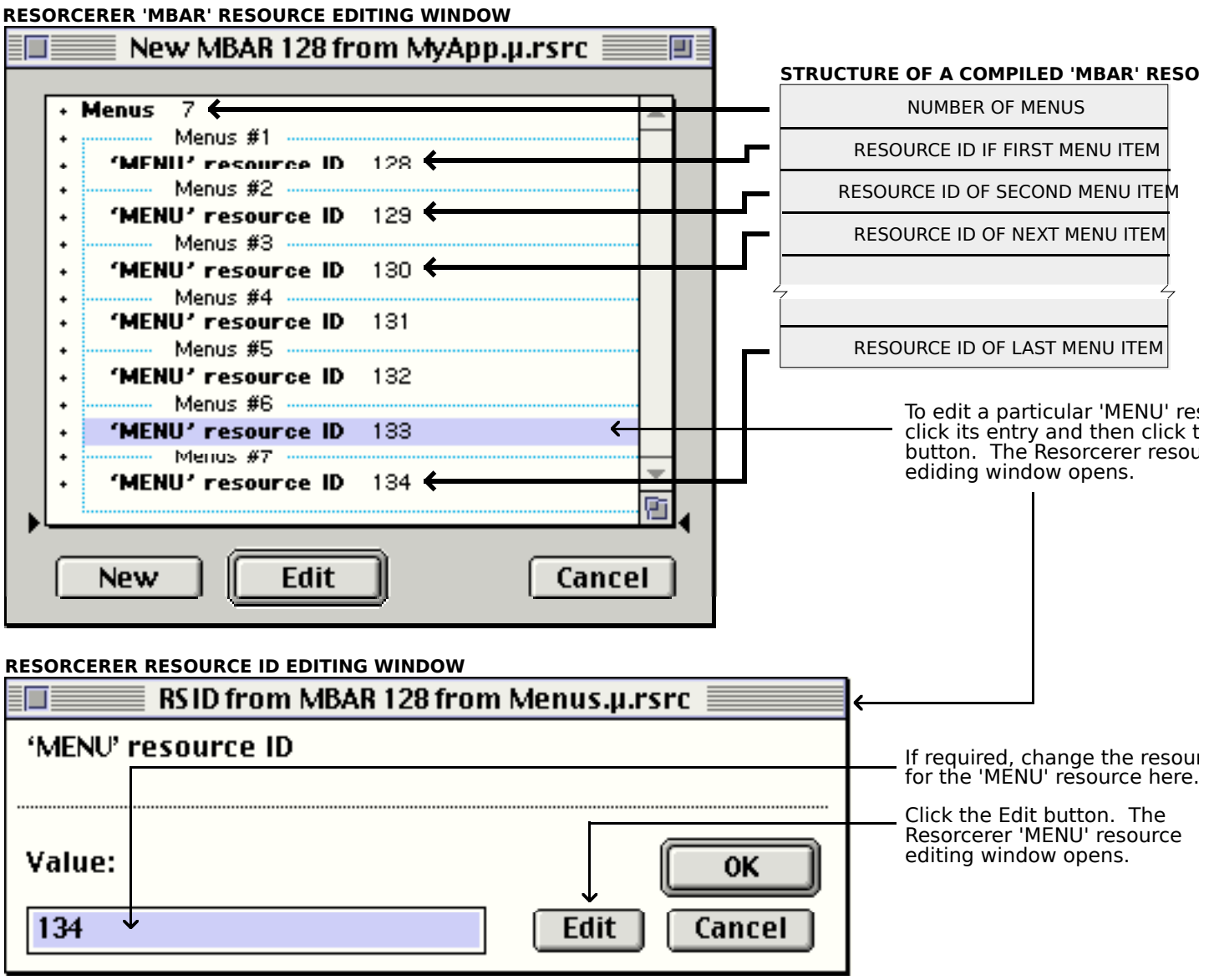

FIG 7 - CREATING AN 'MBAR' RESOURCE USING RESORCERER

Creating 'MENU' Resources

Fig 8 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).

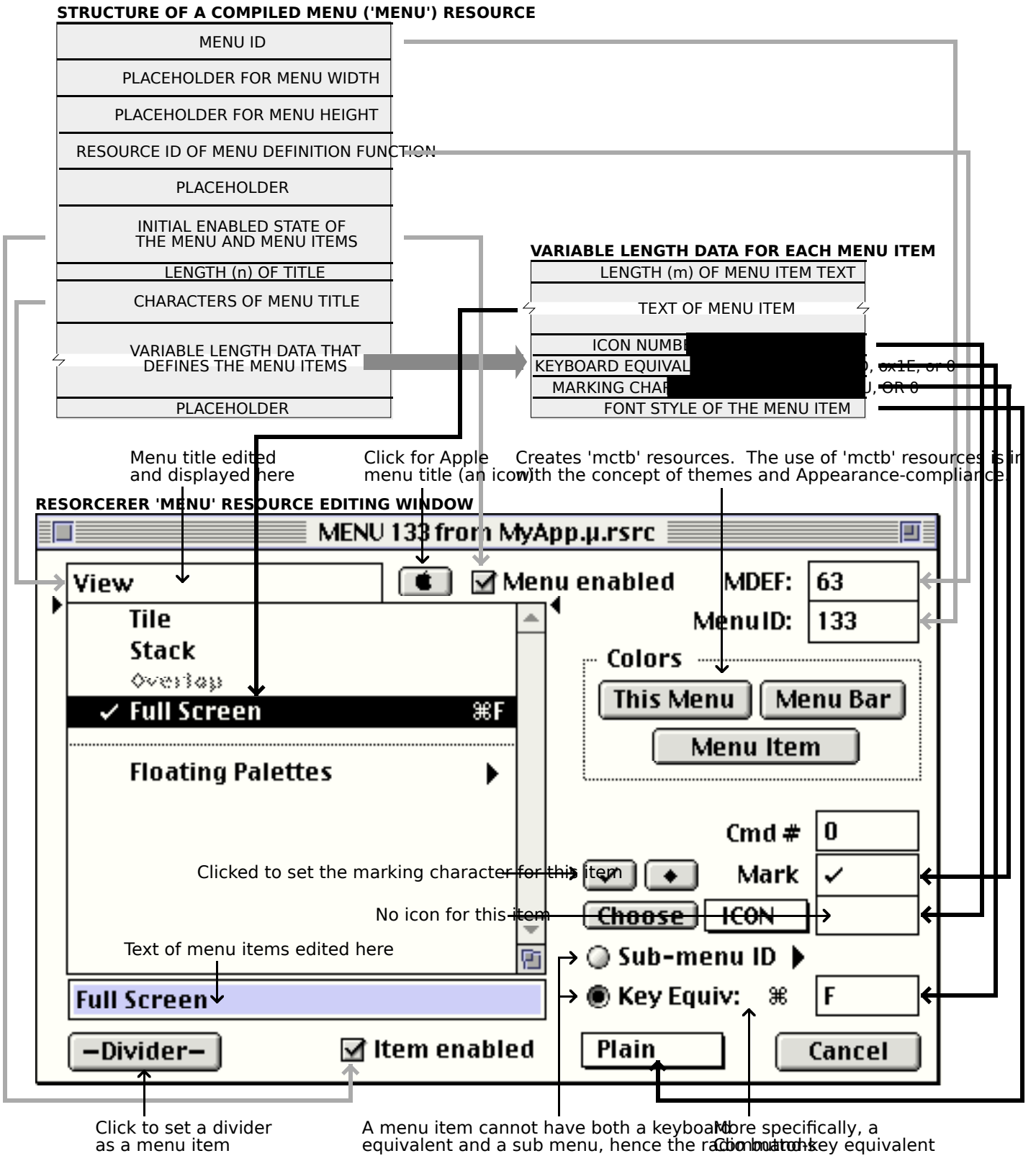

FIG 8 - EDITING A 'MENU' RESOURCE USING RESORCERER

Fig 9 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.

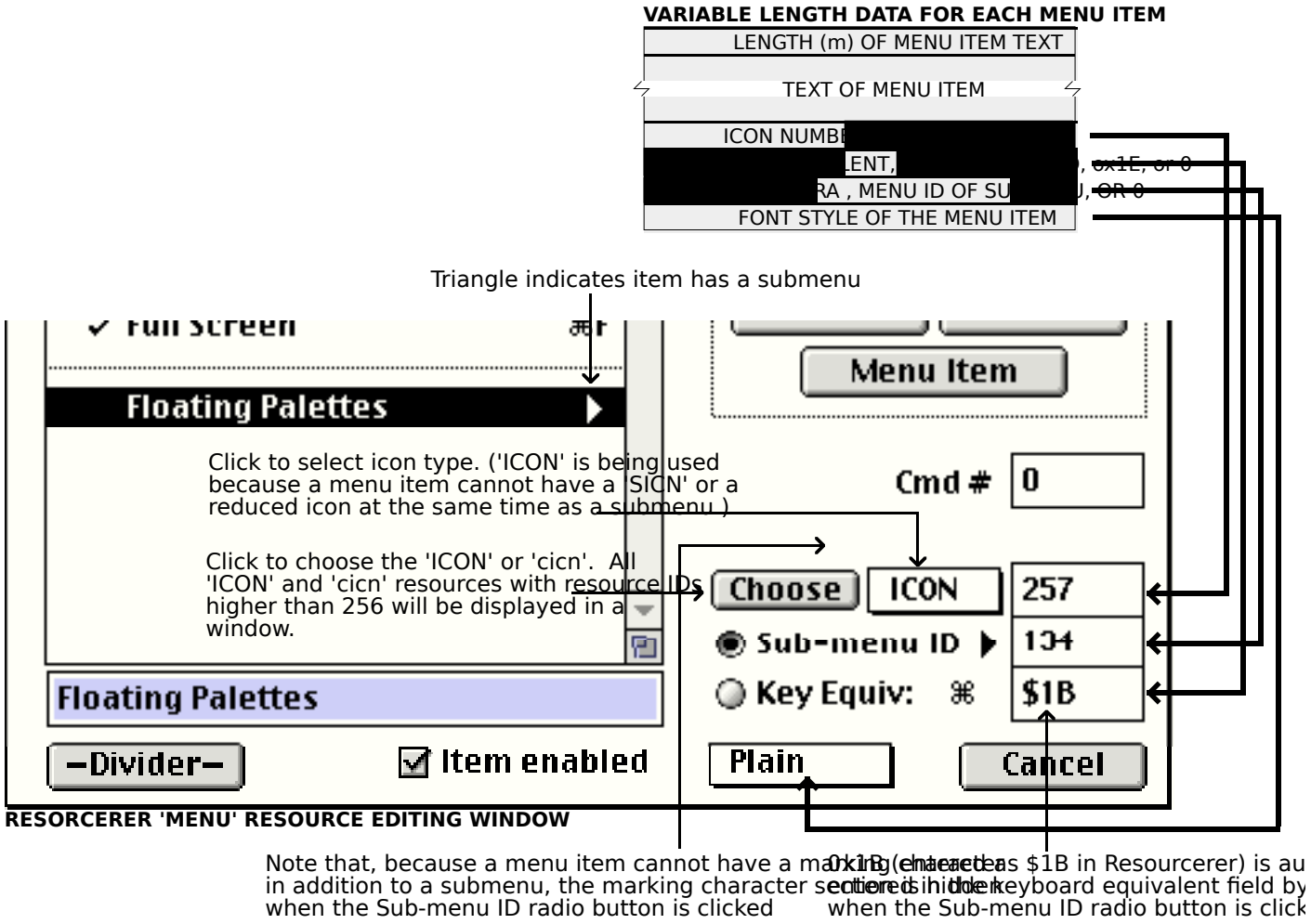

FIG 9 - FURTHER EDITING OF A 'MENU' RESOURCE USING RESORCERER

Creating 'MENU' Resources for Submenus

Fig 10 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

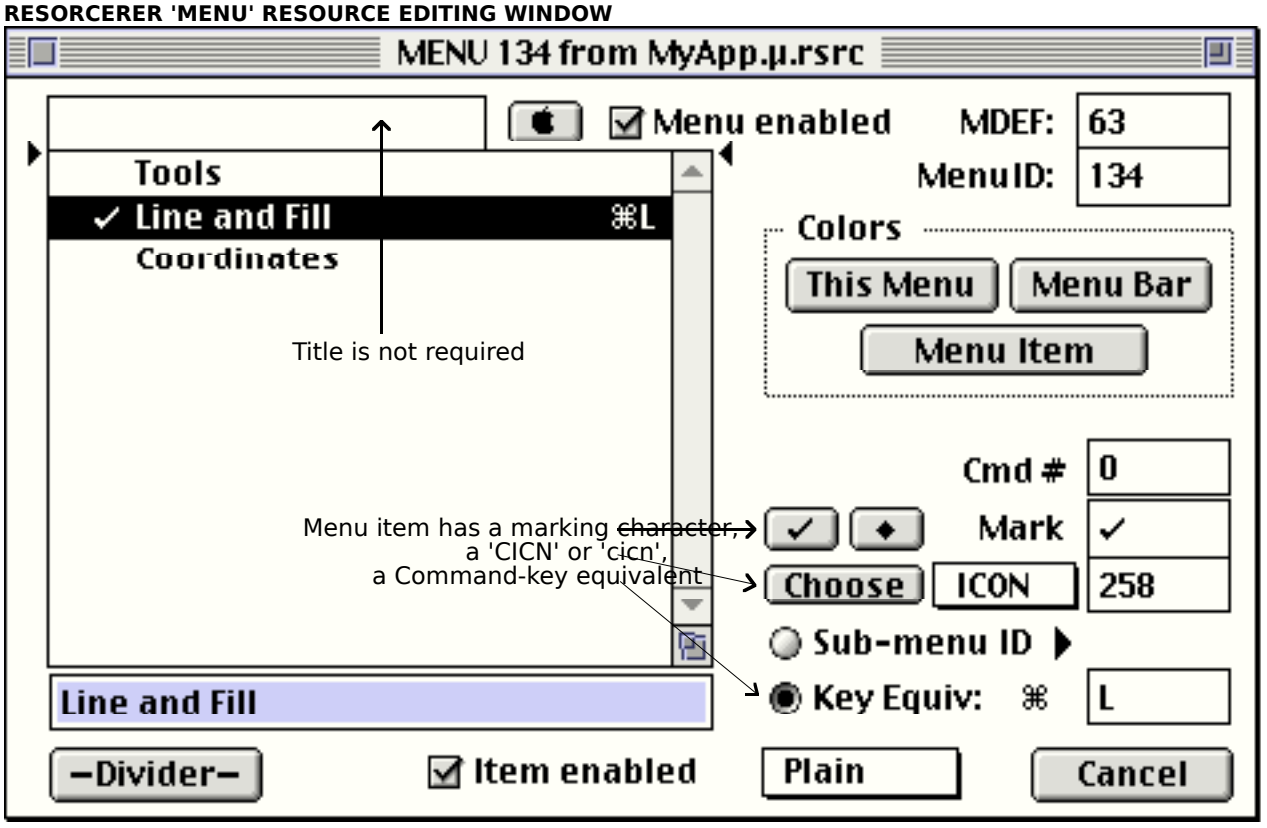

FIG 10 - EDITING A 'MENU' RESOURCE FOR A SUBMENU USING RESORCERER

Creating 'xmnu' Resources

Fig 11 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.

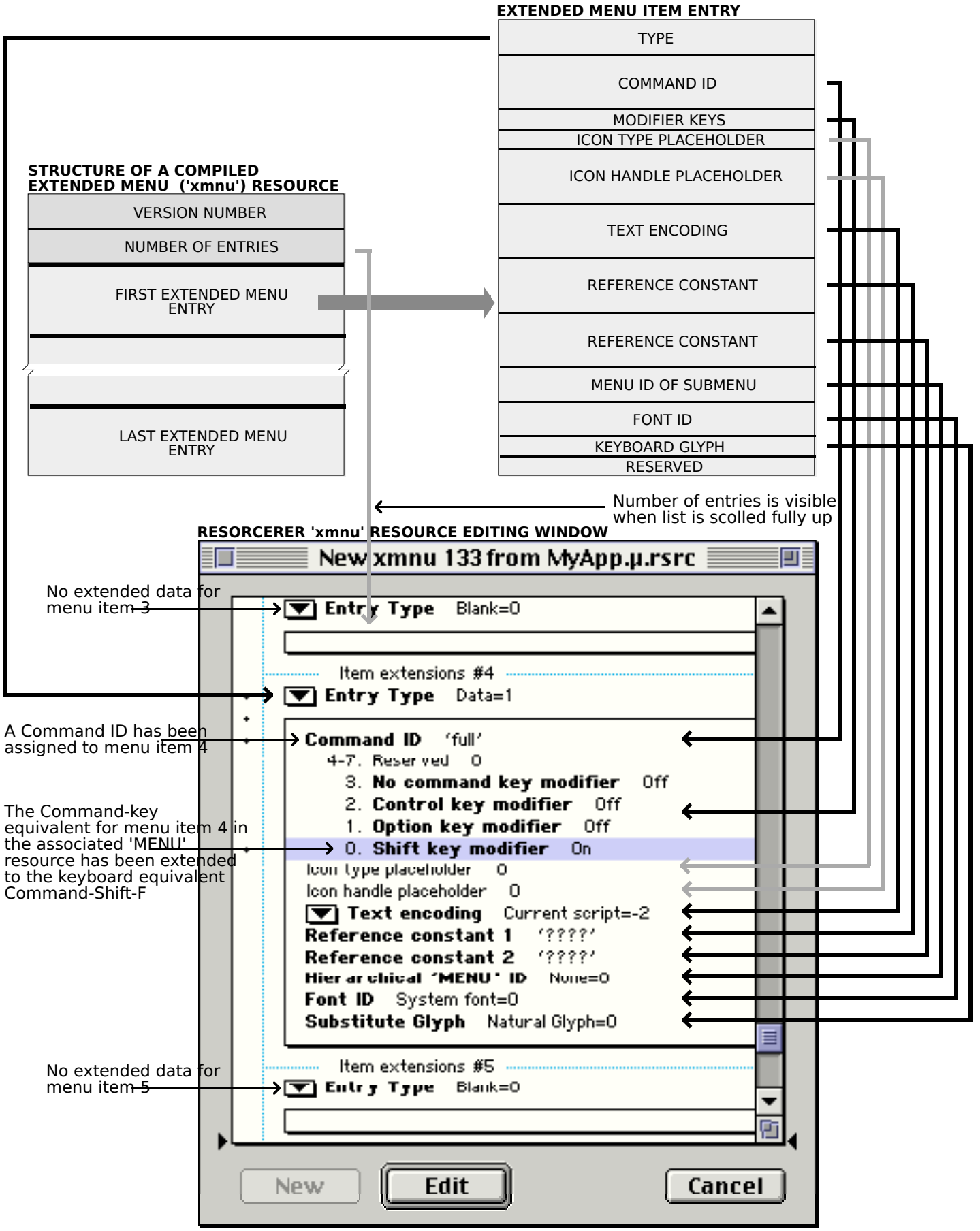

FIG 11 - CREATING AN 'xmnu' RESOURCE USING RESORCERER

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 an 'xmnu' resource with the same resource ID and reads it in if found. GetNewMBar creates a menu structure 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.

Creating an Hierarchical Menu

GetNewMBar does not read in the resource descriptions of submenus but simply records the menu ID of any submenu in the menu structure. 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.[6]

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 GetMenuHandle may be used to obtain a handle to the menu structure of any menu in the current menu list.

Providing Help Balloons

'hmmu' Resources

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. In the case of the 'hmnu' resource for your Help menu item, the resource ID should be -16490.

Creating 'hmnu' Resources

Fig 12 shows an 'hmnu' (help menu) resource being created using Resorcerer.

6 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.

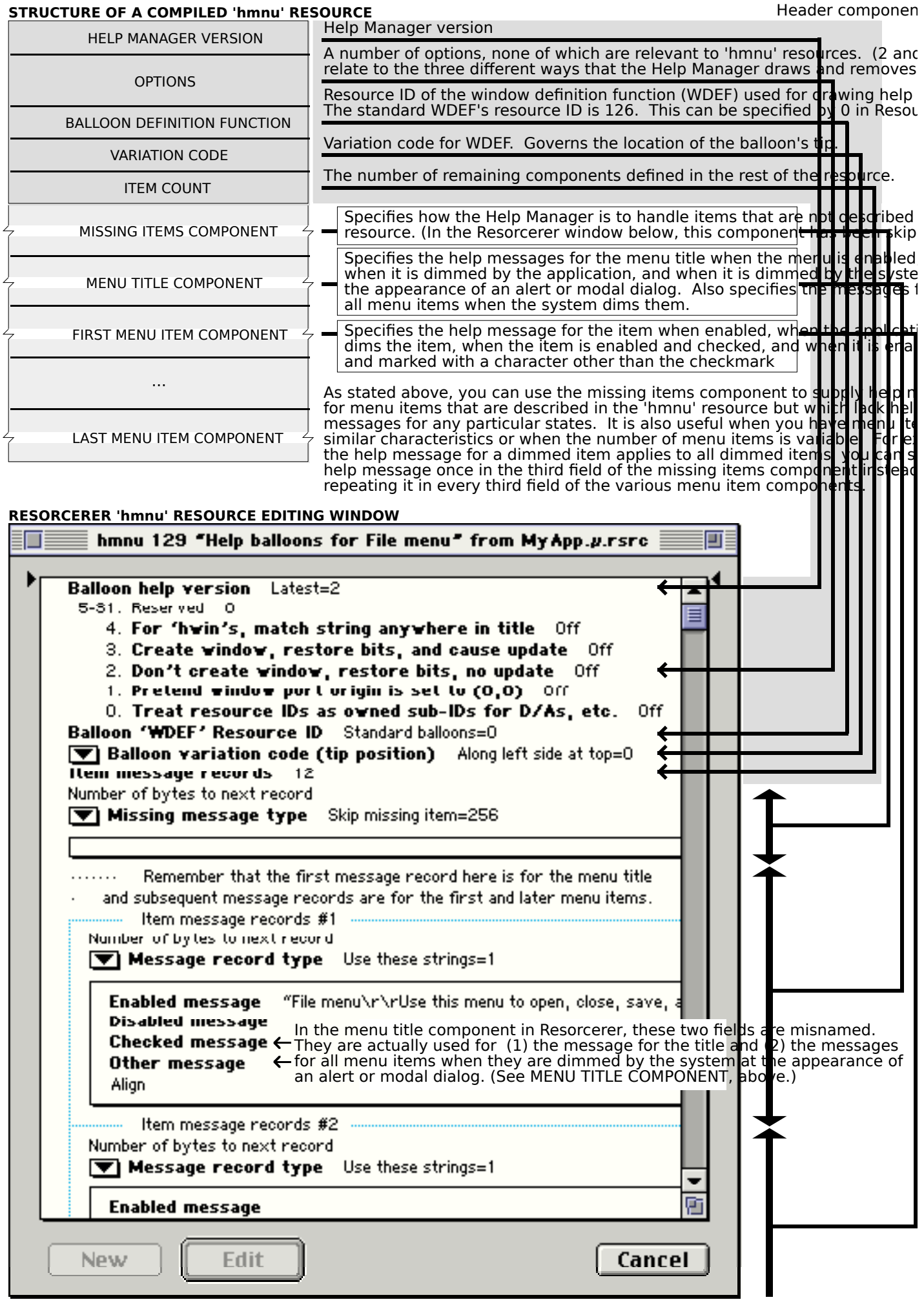

FIG 12 - CREATING AN 'hmnu' RESOURCE USING RESORCERER

Specifying the Format of Help Messages. The example at Fig 12 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 which

facilitates the choice of other formats (and also provides an option which 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.

Changing the Appearance of Items in a Menu

Menu Manager functions may be used to change the appearance of items in a menu, for example, the font style, text or other characteristics. Most of the functions which get or set menu characteristics require three parameters:

- A handle to the menu structure of the menu containing the desired item.

- The number of the menu item.

- A variable which either specifies the data to set or identifies where to return information about that item.

Enabling and Disabling Menu Items

Specific menu items or entire menus are disabled and enabled using:

- DisableItem and EnableItem, which both take a handle to the menu structure 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. DisableItem and EnableItem can only disable and enable menu items with item numbers up to 31.

- In Mac OS 8.5 and later, DisableMenuItem and EnableMenuItem. Unlike DisableItem and EnableItem, DisableMenuItem and EnableMenuItem can enable and disable menu items with item numbers greater than 31.

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 which 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.

Other Appearance Changes

The following lists other functions related to changing the appearance of menu items. Those appearing on a gray background were introduced with Mac OS 8 and the Appearance Manager. Those appearing on a dark gray background were introduced with Mac OS 8.5.

Function	Description
SetMenuItemText GetMenuItemText	Set and get the text.
SetItemStyle GetItemStyle	Set and get the font style.
SetItemMark GetItemMark	Set and get the marking character.
SetItemIcon 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 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 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 and the Appearance Manager.
SetMenuItemKeyGlyph 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 GetMenuFont	Set and get the font used in an individual menu
SetMenuExcludesMarkColumn SetMenuExcludesMarkColumn	Set and determine 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, and InsertMenuItemText (the latter two were introduced with Mac OS 8.5) are used to add items other than the names of resources (such as font resources) to a previously created menu. They require:

- A handle to the menu structure 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 line. 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. If the keyboard equivalent field contains $1B, 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] When followed by $1B, specifies that this menu item has a submenu.[1] (**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)[7], 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 handle. The text for the menu item is "Pick a Colour..." as stored in the preceding 'STR#' resource.

```
var
myMenu : MenuHandle;
itemString: Str255;
...
myMenu := GetMenuHandle(mLibrary);
GetIndString(itemString, 300, 1);
AppendMenu(myMenu, itemString);
```

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

```
var
myMenu : MenuHandle;
itemString: Str255;
...
myMenu := GetMenuHandle(mColours);
GetIndString(itemString, 300, 2);
InsertMenuItem(myMenu, itemString, iRed);
```

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

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

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

The following code adds a divider to the Edit menu:

```
AppendMenu(myMenu, '(-');
```

Strings Without Metacharacters

The two functions introduced with Mac OS 8.5 (AppendMenuItemText and InsertMenuItemText) append and insert the specified string without evaluating the string for metacharacters. These two 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 font resources in the Fonts folder as menu items in your application's Font menu. Similarly, AppendResMenu can be used to add all of the items from the Apple Menu Items folder to your application's Apple menu (with 'DRVR' specified as the resource type in the call). These are common instances of when you will need to add items not already defined in a 'MENU' resource.

Adding Items to the Help Menu

Items are added to your application's Help menu using AppendMenu or InsertMenuItem.

Handling Menu Choices

Determining the Menu ID and Menu Item — MenuSelect and MenuEvent

When the user presses the mouse button while the cursor is in the menu bar, your application should first adjust its menus (that is, enable or disable menu items and add or remove marks as required) and 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 which did not map to a keyboard equivalent, MenuSelect and MenuEvent return 0 in the high word, the value in the low word being undefined.

> ***Historical Note***
>
> The function MenuEvent was introduced with Mac OS 8 and the Appearance Manager as part of the new support for extended modifier keys. Previously, when only Command-key equivalents were supported, the function MenuKey was used. Technically, either MenuEvent or MenuKey could be used if your menus do not require support for extended modifier keys, though MenuEvent is recommended.

Further Handling - Command IDs Not Used

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

Further Handling - Command IDs Used

Mac OS 8 and the Appearance Manager 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 GetMenuHandle to get the handle to the menu's menu structure. This handle 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 application-defined functions which 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 box containing an editable 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 box, etc.. Accordingly, the application-defined menu adjustment function should first determine which window is the front window. The following are examples of menu adjustment functions:

```
procedure DoAdjustMenus;

    var
    theWindowPtr : WindowPtr;
    windowType : SInt16;

    begin
    theWindowPtr := FrontWindow;
    windowType := DoGetWindowType(theWindowPtr);

    case windowType of

        kMyDocWindow: begin
            DoAdjustFileMenuForDocWindow;
            DoAdjustEditMenuForDocWindow;
```

```
                { Adjust others. }
                end;

            kMyModelessDialogWindow: begin
                DoAdjustMenusForModelessDialogs;
                end;

            kNil: begin
                DoAdjustMenusNoWindows;
                end;

            otherwise begin
                end;
            end;
                { of case statement }

        DrawMenuBar;
        end;
            { of procedure DoAdjustMenus }

procedure DoAdjustFileMenuForDocWindow;

    var
    menuHdl : MenuHandle;

    begin
    menuHdl := GetMenuHandle(mFile);

    EnableItem (menuHdl, iNew);
    EnableItem (menuHdl, iOpen);
    DisableItem(menuHdl, iClose);
    DisableItem(menuHdl, iSave);
    DisableItem(menuHdl, iSaveAs);
    DisableItem(menuHdl, iPageSetup);
    DisableItem(menuHdl, iPrint);
    EnableItem (menuHdl, iQuit);
    end;
        { of procedure DoAdjustFileMenuForDocWindow }
```

Handling Apple Menu Choices

When the user chooses an item in the 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 box. Otherwise, your application should use the GetMenuItemText function to get the menu item text and then call the OpenDeskAcc function, passing the text of the chosen menu item as a parameter.

The OpenDeskAcc function prepares to open the desktop object chosen by the user. For example, if the user chose a document created by the SimpleText application, OpenDeskAcc schedules SimpleText for execution (or prepares to open it if it was not already open) and returns to your application. On your application's next call to WaitNextEvent, your application receives a suspend event and the Process Manager makes SimpleText the foreground process, instructing it to open the chosen document.

Handling Help Menu Choices

Both the MenuSelect and MenuEvent functions return the kHMHelpMenuID constant (-16490) in the high word if the user chooses an appended item from the Help menu. The item number of the appended item is returned in the low word. When the kHMHelpMenuID constant is detected, an application-defined function should be called to respond to the user's choice of a Help menu command. That function must accommodate the fact that Apple reserves the right to change the number of standard items in the Help menu.

Handling a Size Menu

Preamble

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 box 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 application-defined function which handles a user's choice of an item in the Size menu:

```
procedure DoHandleSizeCommand(menuItem : SInt16);

    var
    numItems : SInt16;
    addItem : boolean;
    sizeChosen : SInt32;

    begin
    numItems := CountMItems(GetMenuHandle(mSize));
    if (menuItem = numItems) then                    { If user chose Other, display dialog box. If the }
        begin                                        { user-specified size is not in the menu, add a }
        DoDisplayOtherBox(sizeChosen);               { checkmark to the Other command and add the new }
        end                                          { font size to the text of the Other command. }
    else begin    { Return sizeChosen. }
                                                                { User chose a size.  Remove marks }
        DoRemoveMarksFromSizeMenu();                            { from item/s showing previous size. }
        CheckItem(GetMenuHandle(mSize), menuItem,t rue);    { Add mark to chosen item. }
        sizeChosen := DoItemToSize(menuItem);               { Convert item number to font size. }
        end;
    DoResizeSelection(sizeChosen);                   { Update document state or user selection. }
    end;
        { of procedure DoHandleSizeCommand }
```

Hiding and Showing the Menu Bar

Mac OS 8.5 introduced the functions HideMenuBar and ShowMenuBar, which may be used to make the menu bar invisible and unselectable and visible and selectable.

Accessing Menus From Alert and Dialog Boxes

When alert boxes and dialog boxes 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 alert boxes, movable alert boxes, modal dialog boxes, moveable modal dialog boxes, and modeless dialog boxes are displayed is somewhat involved, and is addressed in detail at Chapter 8 — Dialogs and Alerts.

Main Menu Manager Constants, Data Types, and Functions

In the following:

- The constants, data types, and functions introduced with Mac OS 8 and the Appearance Manager are shown on a gray background.
- The functions introduced with Mac OS 8.5 are shown on a dark gray background.
- Those older constants, data types and functions affected by the introduction of Mac OS 8 and the Appearance Manager, but which may still be used in certain circumstances, are shown against a black background.

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
```

Modifier Key Masks for GetMenuItemModifiers and SetMenuItemModifiers Calls

```
kMenuCommandModifiers   = 0     // If no bit is set, only the Command key is used.
kMenuShiftModifier      = $01   // If this bit (bit 0) is set, the Shift key is used.
kMenuOptionModifier     = $02   // If this bit (bit 1) is set, the Option key is used.
kMenuControlModifier    = $04   // If this bit (bit 2) is set, the Control key is used.
kMenuNoCommandModifier  = $08   // 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.
```

Data Types

Menu Structure

```
MenuInfoPtr = ^MenuInfo;
MenuInfo = RECORD
   menuID:          INTEGER;
   menuWidth:       INTEGER;
   menuHeight:      INTEGER;
   menuProc:        Handle;
   enableFlags:     LONGINT;
   menuData:        Str255;
END;

MenuPtr = ^MenuInfo;
MenuHandle = ^MenuPtr;
```

Functions

Initialising the Menu Manager

```
PROCEDURE InitMenus;
```

Creating Menus

```
FUNCTION  NewMenu(menuID: INTEGER; menuTitle: ConstStr255Param): MenuHandle;
FUNCTION  GetMenu(resourceID: INTEGER): MenuHandle;
```

Adding Menus to and Removing Menus From the Current Menu List

```
PROCEDURE InsertMenu(theMenu: MenuHandle; beforeID: INTEGER);
PROCEDURE DeleteMenu(menuID: INTEGER);
PROCEDURE ClearMenuBar;
```

Getting a MenuBar Description From an 'MBAR' resource

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

Getting and Setting the Menu Bar

```
FUNCTION  GetMenuBar: Handle;
PROCEDURE SetMenuBar(menuList: Handle);
FUNCTION GetMBarHeight: INTEGER;
```

Drawing the Menu Bar

```
PROCEDURE DrawMenuBar;
PROCEDURE InvalMenuBar;
```

Controlling Menu Bar Visibility

```
PROCEDURE ShowMenuBar;
PROCEDURE HideMenuBar;
FUNCTION  IsMenuBarVisible: BOOLEAN;
```

Modifying the Menu Width

```
FUNCTION  GetMenuExcludesMarkColumn(menu: MenuHandle): BOOLEAN;
FUNCTION  SetMenuExcludesMarkColumn(menu: MenuHandle; excludesMark: BOOLEAN): OSStatus;
```

Responding to User Choice of a Menu Command

```
FUNCTION  MenuKey(ch: CharParameter): LONGINT;
FUNCTION  MenuEvent({CONST}VAR inEvent: EventRecord): UInt32;
FUNCTION  MenuSelect(startPt: Point): LONGINT;
FUNCTION  MenuChoice: LONGINT;
PROCEDURE HiliteMenu(menuID: INTEGER);
FUNCTION  PopUpMenuSelect(menu: MenuHandle; top: INTEGER; left: INTEGER;
                popUpItem: INTEGER): LONGINT;
```

Getting a Handle to a Menu Structure

```
FUNCTION  GetMenuHandle(menuID: INTEGER): MenuHandle;
FUNCTION  HMGetHelpMenuHandle(VAR mh: MenuHandle): OSErr;
```

Adding and Deleting Menu Items

```
PROCEDURE AppendMenu(menu: MenuHandle; data: ConstStr255Param);
PROCEDURE InsertMenuItem(theMenu: MenuHandle; itemString: ConstStr255Param;
                afterItem: INTEGER);
FUNCTION AppendMenuItemText(menu: MenuHandle; inString: Str255): OSStatus;
FUNCTION InsertMenuItemText(menu: MenuHandle; inString: Str255; afterItem: UInt16): OSStatus;
PROCEDURE DeleteMenuItem(theMenu: MenuHandle; item: INTEGER);
PROCEDURE AppendResMenu(theMenu: MenuHandle; theType: ResType);
PROCEDURE InsertResMenu(theMenu: MenuHandle; theType: ResType; afterItem: INTEGER);
```

Manipulating and Accessing Menu Item Characteristics

```
PROCEDURE EnableItem(theMenu: MenuHandle; item: INTEGER);
```

```
PROCEDURE DisableItem(theMenu: MenuHandle; item: INTEGER);
PROCEDURE EnableMenuItem(theMenu: MenuHandle; item: UInt16);
PROCEDURE DisableMenuItem(theMenu: MenuHandle; item: UInt16);
FUNCTION  IsMenuItemEnabled(menu: MenuHandle; item: UInt16): BOOLEAN;
PROCEDURE EnableMenuItemIcon(theMenu: MenuHandle; item: UInt16);
PROCEDURE DisableMenuItemIcon(theMenu: MenuHandle; item: UInt16);
FUNCTION  IsMenuItemIconEnabled(menu: MenuHandle; item: UInt16): BOOLEAN;
PROCEDURE GetMenuItemText(theMenu: MenuHandle; item: INTEGER; VAR itemString: Str255);
PROCEDURE SetMenuItemText(theMenu: MenuHandle; item: INTEGER; itemString: ConstStr255Param);
PROCEDURE GetItemStyle(theMenu: MenuHandle; item: INTEGER; VAR chStyle: Style);
PROCEDURE SetItemStyle(theMenu: MenuHandle; item: INTEGER; chStyle: StyleParameter);
PROCEDURE GetItemMark(theMenu: MenuHandle; item: INTEGER; VAR markChar: CharParameter);
PROCEDURE SetItemMark(theMenu: MenuHandle; item: INTEGER; markChar: CharParameter);
PROCEDURE CheckItem(theMenu: MenuHandle; item: INTEGER; checked: BOOLEAN);
FUNCTION  GetMenuFont(menu: MenuHandle; VAR outFontID: SInt16;
              VAR outFontSize: UInt16): OSStatus;
FUNCTION  SetMenuFont(menu: MenuHandle; inFontID: SInt16; inFontSize: UInt16): OSStatus;
PROCEDURE GetItemIcon(theMenu: MenuHandle; item: INTEGER; VAR iconIndex: Byte);
PROCEDURE SetItemIcon(theMenu: MenuHandle; item: INTEGER; iconIndex: INTEGER);
PROCEDURE GetItemCmd(theMenu: MenuHandle; item: INTEGER; VAR cmdChar: CharParameter);
PROCEDURE SetItemCmd(theMenu: MenuHandle; item: INTEGER; cmdChar: CharParameter);
FUNCTION  SetMenuItemCommandID(inMenu: MenuHandle; inItem: SInt16;
                  inCommandID: UInt32): OSErr;
FUNCTION  GetMenuItemCommandID(inMenu: MenuHandle; inItem: SInt16;
                  VAR outCommandID: UInt32): OSErr;
FUNCTION  SetMenuItemFontID(inMenu: MenuHandle; inItem: SInt16; inFontID: SInt16): OSErr;
FUNCTION  GetMenuItemFontID(inMenu: MenuHandle; inItem: SInt16; VAR outFontID: SInt16): OSErr;
FUNCTION  SetMenuItemHierarchicalID(inMenu: MenuHandle; inItem: SInt16;
                     inHierID: SInt16): OSErr;
FUNCTION  GetMenuItemHierarchicalID(inMenu: MenuHandle; inItem: SInt16;
                     VAR outHierID: SInt16): OSErr;
FUNCTION  SetMenuItemIconHandle(inMenu: MenuHandle; inItem: SInt16; inIconType: ByteParameter;
                   inIconHandle: Handle): OSErr;
FUNCTION  GetMenuItemIconHandle(inMenu: MenuHandle; inItem: SInt16; VAR outIconType: UInt8;
                   VAR outIconHandle: Handle): OSErr;
FUNCTION  SetMenuItemKeyGlyph(inMenu: MenuHandle; inItem: SInt16; inGlyph: SInt16): OSErr;
FUNCTION  GetMenuItemKeyGlyph(inMenu: MenuHandle; inItem: SInt16;
                  VAR outGlyph: SInt16): OSErr;
FUNCTION  SetMenuItemModifiers(inMenu: MenuHandle; inItem: SInt16;
                  inModifiers: ByteParameter): OSErr;
FUNCTION  GetMenuItemModifiers(inMenu: MenuHandle; inItem: SInt16;
                     VAR outModifiers: UInt8): OSErr;
FUNCTION  SetMenuItemRefCon(inMenu: MenuHandle; inItem: SInt16; inRefCon: UInt32): OSErr;
FUNCTION  GetMenuItemRefCon(inMenu: MenuHandle; inItem: SInt16; VAR outRefCon: UInt32): OSErr;
FUNCTION  SetMenuItemRefCon2(inMenu: MenuHandle; inItem: SInt16; inRefCon2: UInt32): OSErr;
FUNCTION  GetMenuItemRefCon2(inMenu: MenuHandle; inItem: SInt16;
                  VAR outRefCon2: UInt32): OSErr;
FUNCTION  SetMenuItemTextEncoding(inMenu: MenuHandle; inItem: SInt16;
                    inScriptID: TextEncoding): OSErr;
FUNCTION  GetMenuItemTextEncoding(inMenu: MenuHandle; inItem: SInt16;
                    VAR outScriptID: TextEncoding): OSErr;
```

Disposing of Menus

```
PROCEDURE DisposeMenu(theMenu: MenuHandle);
```

Counting Items in a Menu

```
FUNCTION  CountMItems(theMenu: MenuHandle): INTEGER;
```

Highlighting the Menu Bar

```
PROCEDURE FlashMenuBar(menuID: INTEGER);
PROCEDURE SetMenuFlash(count: INTEGER);
```

Recalculating Menu Dimensions

```
PROCEDURE CalcMenuSize(theMenu: MenuHandle);
```

Demonstration Program Menus 1

```
{  ◊◊◊
```

```
// This program:
//
// •   Opens a window.
//
// •   Creates these pull-down menus: Apple, File, Edit, Font, Style, Size, and Special.
//
//     The Apple menu includes an 'About...' menu item for the program.
//
//     The second menu item in the Special menu contains a submenu.
//
//     A 'Help' menu item for the program is appended to the Help menu.
//
// •   Displays text in the window indicating the menu selection made by the user.
//
// 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.
//
// To facilitate a comparison with the fully Appearance-compliant menus in the
// demonstration program Menus2, no measures are taken in this program to cause the menu
// bar and menus to use the new menu bar and menu definition functions when system-wide
// Appearance is selected off in the Appearance control panel.  The menu bar and menus will
// only use the new definition functions when system-wide Appearance is selected on.
//
// 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.
//
// The program utilises the following resources:
//
// •   A 'WIND' resource (purgeable) (initially not visible).
//
// •   An 'MBAR' resource (preload, non-purgeable).
//
// •   'MENU' resources for the drop-down and hierarchical menus (all preload, all
//     non-purgeable).
//
//
// ◊◊◊
```

```
 iTen = 1;
 iTwelve = 2;
 iEighteen = 3;
 iTwentyFour = 4;
mSpecial = 134;
 iFirstItem = 1;
 iSecondItem = 2;
mSubmenu = 135;
 iFirstSubItem = 1;
 iSecondSubItem = 2;
rWindowResource = 128;

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

var

gDone : Boolean;
gCurrentFont : SInt16;
gCurrentStyle : Style;
gCurrentSize : SInt16;
gEvent : EventRecord;
gWindowPtr : WindowPtr;

//
..........................................................................................................................................
............ routine prototypes

procedure DoInitManagers; forward;
procedure DoGetMenus; forward;
procedure DoEvents({$const} var theEvent : EventRecord); forward;
procedure DoMouseDown({$const} var theEvent : EventRecord); forward;
procedure DoAdjustMenus; forward;
procedure DoMenuChoice(menuChoice : SInt32); forward;
procedure DoAppleMenu(theItem : SInt16); forward;
procedure DoFileMenu(theItem : SInt16); forward;
procedure DoEditMenu(theItem : SInt16); forward;
procedure DoFontMenu(theItem : SInt16); forward;
procedure DoStyleMenu(theItem : SInt16); forward;
procedure DoSizeMenu(theItem : SInt16); forward;
procedure DoSpecialMenu(theItem : SInt16); forward;
procedure DoSubMenus(theItem : SInt16); forward;
procedure DoHelpMenu(theItem : SInt16); forward;
procedure DrawItemString(theString : Str255); forward;

// ◊◊◊
```

```
DrawMenuBar;

menuHdl := GetMenuHandle(mApple);
if (menuHdl <> nil) then
    begin
    AppendResMenu(menuHdl,'DRVR');
    end
else begin
    ExitToShell;
    end;

menuHdl := GetMenuHandle(mFont);
if (menuHdl <> nil) then
    begin
    AppendResMenu(menuHdl,'FONT');
    end
else begin
    ExitToShell;
    end;

menuHdl := GetMenu(mSubmenu);
if (menuHdl <> nil) then
    begin
    InsertMenu(menuHdl,hierMenu);
    end
else begin
    ExitToShell;
    end;

osErr := HMGetHelpMenuHandle(menuHdl);
if (osErr = noErr) then
    begin
    AppendMenu(menuHdl,'Menus1 Help');
    end
else begin
    ExitToShell;
    end;

DoFontMenu(gCurrentFont);
DoStyleMenu(0);
    DoSizeMenu(gCurrentSize);
    end;
        { of procedure DoGetMenus }

// ◊◊◊
```

```
        { of procedure DoEvents }

// ◊◊◊
```

```
            end;

        mFont: begin
            DoFontMenu(menuItem);
            end;

        mStyle: begin
            DoStyleMenu(menuItem);
            end;

        mSize: begin
            DoSizeMenu(menuItem);
            end;

        mSpecial: begin
            DoSpecialMenu(menuItem);
            end;

        mSubmenu: begin
            DoSubMenus(menuItem);
            end;

        kHMHelpMenuID: begin
            DoHelpMenu(menuItem);
            end;

        otherwise begin
            end;
        end;
            { of case statement }

    HiliteMenu(0);
    end;
        { of procedure DoMenuChoice }

// ◊◊◊
```

```
            DrawItemString('Copy');
            end;

        iPaste: begin
            DrawItemString('Paste');
            end;

        iClear: begin
            DrawItemString('\Clear');
            end;

        otherwise begin
            end;
        end;
            { of case statement }
    end;
        { of procedure DoEditMenu }

// ◊◊◊
```

```
            else begin
                gCurrentStyle := gCurrentStyle + [underline];
                end;
            end;

        iOutline: begin
            if (outline in gCurrentStyle) then
                begin
                gCurrentStyle := gCurrentStyle - [outline];
                end
            else begin
                gCurrentStyle := gCurrentStyle + [outline];
                end;
            end;

        iShadow: begin
            if (shadow in gCurrentStyle) then
                begin
                gCurrentStyle := gCurrentStyle - [shadow];
                end
            else begin
                gCurrentStyle := gCurrentStyle + [shadow];
                end;
            end;

        otherwise begin
            end;

        end;
            { of case statement }

    styleMenuHdl := GetMenuHandle(mStyle);

    CheckItem(styleMenuHdl, iPlain, gCurrentStyle = []);
    CheckItem(styleMenuHdl, iBold, bold in gCurrentStyle);
    CheckItem(styleMenuHdl, iItalic, italic in gCurrentStyle);
    CheckItem(styleMenuHdl, iUnderline, underline in gCurrentStyle);
    CheckItem(styleMenuHdl, iOutline, outline in gCurrentStyle);
    CheckItem(styleMenuHdl, iShadow, shadow in gCurrentStyle);

    TextFace(gCurrentStyle);

    DrawItemString('Style change');
    end;
        { of procedure DoStyleMenu }

// ◊◊◊
```

```
    gCurrentSize := menuItem;

    DrawItemString('Size change');
    end;
        { of procedure DoSizeMenu }

// ◊◊◊
```

```
        DrawString(eventString);
        end;
    end;
        { of procedure DrawItemString }

// ◊◊◊
```

The main program block initialises the system software managers, creates a window and makes its graphics port the current port, calls the application-defined function which sets up the menus, shows the window and enters 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 record 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 and DrawMenuBar draws the menu bar.

The next block adds the contents of the Apple Menu Items folder to the Apple menu. The use of 'DRVR' as the second parameter to the AppendResMenu call is automatically interpreted to mean that the Apple menu is being created, so that all items in the Apple Menu Items folder are added rather than resources of type 'DRVR'.

The third block adds the names of all resident fonts to the Font menu. Using 'FONT' in the second parameter in the call to AppendResMenu causes all such resources to be searched out and their names added to the specified menu.

The fourth block inserts the application's single submenu 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 record 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 next block appends a menu item with the name "Menus Help" to the Help menu.

The last three lines cause checkmarks to be set against the appropriate font, style and size menu items according to the initialised values of the associated global variables.

DoEvents

DoEvents branches 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.

In the case of key-down and auto-key events, the character code is first extracted from the event record's message field. A check is then made of the modifiers field 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.

The call to HiliteMenu at the osEvt case unhighlights the Apple menu title when the user brings the demonstration program to the foreground, having previously sent it to the background by choosing an Apple Menu Items folder item from the Apple menu.

DoMouseDown

DoMouseDown first establishes the window and window part in which the mouse-down event occurred, and branches 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.

The inDrag case responds to a mouse-down in the drag bar.

The inGoAway case responds to a mouse-down in the go-away box, setting gDone to true and thus terminating the program if the cursor is still within the go-away box when the mouse button is released.

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 taken in this simple program because only one window, whose content never changes, is ever open.

(Later demonstration programs contain examples of menu adjustment functions which cater for specific circumstances. For example, the menu adjustment function in the demonstration program at Chapter 8 — Dialogs and Alerts accommodates the situation where the front window could be either a document window or a modeless dialog box.)

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).

Within the switch on the menu ID, the appropriate application-defined individual menu handling function are called. Note the handling of the hierarchical menu (case mSubMenu). Note also that, at the last case, the kHMHelpMenuID constant (-16490) is returned in the high word if the user chooses an appended item from the Help menu.

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

DoAppleMenu

DoAppleMenu takes the short integer representing the menu item. If this value represents the first item in the Apple menu (the inserted "About..." item), text representing this item is drawn in the scrolling display.

If the value passed to the DoAppleMenu routine represents other items in the Apple menu, the call to GetMenuItemText gets the string representing the item's name. This string (which excludes metacharacters) is used as the parameter in the OpenDeskAcc call. OpenDeskAcc opens the chosen object and passes control that object.

DoFileMenu

DoFileMenu handles choices from the File menu. 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 branches according to the menu item number, drawing text representing the chosen item in the window.

DoFontMenu

DoFontMenu first gets a handle to the Font menu structure required by the following CheckItem calls. The CheckItem calls uncheck the current font menu item and check the menu item passed to the DoFontMenu function. This latter menu item number is then assigned to the gCurrentFont global variable.

The call to GetMenuItemText extracts the string representing the item's name. This string is passed as the first parameter in the call to GetFNum, which gets the font number associated with the name. This number is then used in the call to TextFont, which will cause subsequent text drawing to be conducted in the specified font. The last line draws the name of the font in that font.

DoStyleMenu

DoStyleMenu branches according to the menu item chosen in the Style menu. Within the case statement, items in the global set variable gCurrentStyle are added or removed 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 remove all members from 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 the appropriate memberships of gCurrentStyle attended to, a handle to the Style menu record is then obtained. This is required for the six CheckMenuItem, which check or uncheck the individual menu items according to whether the third argument 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.

DoSizeMenu

DoSizeMenu branches 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 selected 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 branches according to the chosen subitem in the hierarchical menu represented by the second menu item in the Special menu.

DoHelpMenu

DoHelpMenu handles the choice of the "Menus Help" item added by this program to the system-managed Help Menu. This code reflects the fact that Apple reserves the right to add items to the Help menu in future versions of the system software.

HMGetHelpMenuHandle gets a handle to the Help menu record. The call to CountMenuItems returns the number of items in the Help menu. Since we know that we have added one item to this menu, the next line will establish the original number of help items. If the value passed to the DoHelpMenu routine is greater than this number, it must therefore represent the item number of our "Menus Help" item, in which case some text is drawn in the window to register the fact.

DrawItemString

The routine 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 Menus 2

```
{  ◊◊◊
```

```
//
// 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 via calls to SetMenuItemModifiers.
//
// 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 '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.
//
// ◊◊◊
```

```
mSize = 134;
 iTen = 1;
 iTwelve = 2;
 iEighteen = 3;
 iTwentyFour = 4;
 idTen = 'ten ';
 idTwelve = 'twel';
 idEighteen = 'eigh';
 idTwentyFour = 'twen';
mSpecial = 135;
 iFirst = 1;
 iSecond = 2;
 idFirst = 'firs';
mSubmenu = 136;
 iBat = 1;
 iBowl = 2;
 idBat = 'bat ';
 idBowl = 'bowl';
rWindowResource = 128;
rColourIcon = 258;

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

var

gMacOS_85_present : Boolean;
gDone : Boolean;
gCurrentFont : SInt16;
gCurrentStyle : Style;
gCurrentSize : SInt16;
gEvent : EventRecord;
gWindowPtr : WindowPtr;
backColour : RGBColor;
foreColour : RGBColor;
gIgnoredErr : OSErr;

//
..........
... main program variables

osError : OSErr;
response : SInt32;

//
..........
..... routine prototypes

procedure DoInitManagers; forward;
procedure DoGetMenus; forward;
procedure DoEvents({$const} var theEvent : EventRecord); forward;
procedure DoMouseDown({$const} var theEvent : EventRecord); forward;
procedure DoAdjustMenus; forward;
procedure DoMenuChoice(menuChoice : SInt32); forward;
procedure DoAppleMenu(menuItem : SInt16); forward;
procedure DoFontMenu(menuItem : SInt16); forward;
procedure DoHelpMenu(menuItem : SInt16); forward;
procedure DoCheckStyleMenuItem(menuID : SInt16); forward;
procedure DoCheckSizeMenuItem(menuID, menuItem : SInt16); forward;
procedure DrawItemString(theString : Str255); forward;

// ◊◊◊
```

```
// ◊◊◊
```

```
    menuHdl := GetMenuHandle(mStyle_prog);
    ignoredErr := SetMenuItemCommandID(menuHdl, iPlain, UInt32(idPlain_prog));
    ignoredErr := SetMenuItemCommandID(menuHdl, iBold, UInt32(idBold_prog));
    ignoredErr := SetMenuItemCommandID(menuHdl, iItalic, UInt32(idItalic_prog));
    ignoredErr := SetMenuItemCommandID(menuHdl, iUnderline, UInt32(idUnderline_prog));
    ignoredErr := SetMenuItemCommandID(menuHdl, iOutline, UInt32(idOutline_prog));
    ignoredErr := SetMenuItemCommandID(menuHdl, iShadow, UInt32(idShadow_prog));

    menuHdl := GetMenuHandle(mSize);
    ignoredErr := SetMenuItemCommandID(menuHdl, iTen, UInt32(idTen));
    ignoredErr := SetMenuItemCommandID(menuHdl, iTwelve, UInt32(idTwelve));
    ignoredErr := SetMenuItemCommandID(menuHdl, iEighteen, UInt32(idEighteen));
    ignoredErr := SetMenuItemCommandID(menuHdl, iTwentyFour, UInt32(idTwentyFour));

    menuHdl := GetMenuHandle(mSpecial);
    ignoredErr := SetMenuItemCommandID(menuHdl, iFirst, UInt32(idFirst));

    menuHdl := GetMenuHandle(mSubmenu);
    ignoredErr := SetMenuItemCommandID(menuHdl, iBat, UInt32(idBat));
    ignoredErr := SetMenuItemCommandID(menuHdl, iBowl, UInt32(idBowl));

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

    cicnHdl := GetCIcon(rColourIcon);
    ignoredErr := SetMenuItemIconHandle(menuHdl, iBowl, kMenuColorIconType, Handle(cicnHdl));

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

    menuHdl := GetMenuHandle(mSize);
    SetItemCmd(menuHdl, iTen, char($08));
    ignoredErr := SetMenuItemKeyGlyph(menuHdl, iTen, $17);
    SetItemCmd(menuHdl, iTwelve, char($7f));
    ignoredErr := SetMenuItemKeyGlyph(menuHdl, iTwelve, $0A);
    SetItemCmd(menuHdl, iEighteen, char($0b));
    SetItemCmd(menuHdl, iTwentyFour, char($0c));

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

{$ifc TARGET_CPU_PPC}
    if(gMacOS_85_present) then
        begin
        menuHdl := GetMenuHandle(mSpecial);
        ignoredErr := SetMenuExcludesMarkColumn(menuHdl,true);

        GetFNum('Gadget',fontNumber);
        if(fontNumber <> 0) then
            begin
            ignoredErr := SetMenuFont(menuHdl,fontNumber,12);
            end
        end;
{$endc}

    //
.....
... set up the Help menu

    osErr := HMGetHelpMenuHandle(menuHdl);
    if (osErr = noErr) then
        begin
        AppendMenu(menuHdl, 'Menus2 Help');
        end
    else begin
        ExitToShell;
        end;

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

    DoFontMenu(gCurrentFont);
    DoCheckStyleMenuItem(mStyle_xmnu);
    DoCheckStyleMenuItem(mStyle_prog);
    DoCheckSizeMenuItem(mSize, iTen);
    end;
        { of procedure DoGetMenus }

// ◊◊◊
```

```
    begin
    case (theEvent.what) of

        mouseDown: begin
            DoMouseDown(theEvent);
            end;

        keyDown, autoKey: begin
            charCode := chr(BAnd(theEvent.message, charCodeMask));
            if (BAnd(theEvent.modifiers, cmdKey) <> 0) then
                begin
                DoAdjustMenus;
                DoMenuChoice(MenuEvent(theEvent));
                end;
            end;

        updateEvt: begin
            BeginUpdate(WindowPtr(theEvent.message));
            EndUpdate(WindowPtr(theEvent.message));
            end;

        osEvt: begin
            HiliteMenu(0);
            end;

        otherwise begin
            end;
        end;
            { of case statement }
    end;
        { of procedure DoEvents }

// ◊◊◊
```

```
    begin
    // Adjust menus here.
    end;
        { of procedure DoAdjustMenus }

// ◊◊◊
```

```
UInt32('bolx'), UInt32('bolp'): begin
    if (bold in gCurrentStyle) then
        begin
        gCurrentStyle := gCurrentStyle - [bold];
        end
    else begin
        gCurrentStyle := gCurrentStyle + [bold];
        end;
    DoCheckStyleMenuItem(menuID);
    end;

UInt32('itax'), UInt32('itap'): begin
    if (italic in gCurrentStyle) then
        begin
        gCurrentStyle := gCurrentStyle - [italic];
        end
    else begin
    gCurrentStyle := gCurrentStyle + [italic];
        end;
    DoCheckStyleMenuItem(menuID);
    end;

UInt32('undx'), UInt32('undp'): begin
    if (underline in gCurrentStyle) then
        begin
        gCurrentStyle := gCurrentStyle - [underline];
        end
    else begin
        gCurrentStyle := gCurrentStyle + [underline];
        end;
    DoCheckStyleMenuItem(menuID);
    end;

UInt32('outx'), UInt32('outp'): begin
    if (outline in gCurrentStyle) then
        begin
        gCurrentStyle := gCurrentStyle - [outline];
        end
    else begin
        gCurrentStyle := gCurrentStyle + [outline];
        end;
    DoCheckStyleMenuItem(menuID);
    end;

UInt32('shax'), UInt32('shap'): begin
    if (shadow in gCurrentStyle) then
        begin
        gCurrentStyle := gCurrentStyle - [shadow];
        end
    else begin
        gCurrentStyle := gCurrentStyle + [shadow];
        end;
    DoCheckStyleMenuItem(menuID);
    end;

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

UInt32('ten '): begin
    TextSize(10);
    DoCheckSizeMenuItem(menuID, menuItem);
    end;

UInt32('twel'): begin
    TextSize(12);
    DoCheckSizeMenuItem(menuID, menuItem);
    end;

UInt32('eigh'): begin
    TextSize(18);
    DoCheckSizeMenuItem(menuID, menuItem);
    end;

UInt32('twen'): begin
    TextSize(24);
    DoCheckSizeMenuItem(menuID, menuItem);
    end;
```

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

                UInt32('firs'): begin
                    DrawItemString('First Item');
                    end;

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

                UInt32('bat '): begin
                    menuHdl := GetMenuHandle(mSubmenu);
                    DisableItem(menuHdl, iBat);
                    EnableItem(menuHdl, iBowl);
                    DrawItemString('Bat');
                    end;

                UInt32('bowl'): begin
                    menuHdl := GetMenuHandle(mSubmenu);
                    DisableItem(menuHdl, iBowl);
                    EnableItem(menuHdl, iBat);
                    DrawItemString('Bowl');
                    end;

                otherwise begin
                    end;
                end;
                { of case statement }
            end;
        end;

    HiliteMenu(0);
    end;
        { of procedure DoMenuChoice }

// ◊◊◊
```

```
// ◊◊◊
```

```
        MoveTo(8,286);
        DrawString(eventString);
        end;
    end;
        { of procedure DrawItemString }

// ◊◊◊
```

Demonstration Program Menus2 Comments

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

- That, if the program is being run under Mac OS 8.0 or 8.1, the appearance of the menu bar and menus remains the same regardless of whether system-wide appearance is selected on or off in the Mac OS 8.0 and 8.1 Appearance control panel.
- 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, if the program is compiled and run under Mac OS 8.5, 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, if the program is being run under Mac OS 8.0 or 8.1, there is an anomaly in the way the new MDEF draws Style menus. In the Style ('xmnu') menu, in which the extended modifier keys for the last two menu items are assigned via the 'xmnu' resource, the item text is severely truncated and an ellipsis is added. In the Style (Programmatic) menu, in which the extended modifier keys are assigned programmatically, this truncation of the item text does not occur. (This problem was fixed in Mac OS 8.5.)
- The balloon help provided for all menus and menu items.

Because this demonstration program is based on Menus1, the following comments exclude those routines which remain unchanged.

constants

Menus2 now establishes constants for command IDs for menu items. Command IDs are of type UInt32. To enhance source code readability, these are defined in the four-character-code format, which packs four one-byte characters together in a 32-bit value.

main program block

The call to RegisterAppearanceClient means that the new Appearance-compliant menu bar definition function (resource ID 63) will be used regardless of whether the system-wide Appearance is selected on or off in the Mac OS 8.0 and 8.1 Appearance control panel.

Next, the function Gestalt is used to determine whether Mac OS 8.5 or later is present. If so, the global variable gMacOS_85_present is set to true. (In the function doGetMenus, this global variable will determine whether two functions introduced with Mac OS 8.5 get called.)

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 structure 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 and DrawMenuBar draws the menu bar.

The next block adds the contents of the Apple Menu Items folder to the Apple menu. The use of 'DRVR' as the second parameter to the AppendResMenu call is automatically interpreted to mean that the Apple menu is being created, so that all items in the Apple Menu Items folder are added rather than resources of type 'DRVR'.

The next block adds the names of all resident fonts to the Font menu and makes the menu WYSIWYG. The call to AppendResMenu causes all 'FONT' resources to be searched out and their names added to the specified menu. The process of making the menu WYSIWYG then begins. The call to CountMItems returns the number of items in the menu. Then, for each of these items, GetMenuItemText gets the text (the font's name), GetFNum gets the font number associated with the font name, and SetMenuItemFontID sets the font for the menu item.

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 structure 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 GetMenuHandle gets the handle 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 this application's items in the system-managed menus (Apple and Help), nor is it possible to assign command IDs to the items in the Font menu.)

The following small 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 second 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 in the case of the two delete keys, otherwise the correct glyphs will not be displayed. The calls to SetItemCmd assign the specified key to the menu item. In the case of the first two calls, a substitute glyph is assigned via calls to SetMenuItemGlyph. If this is not done, the glyphs displayed will not be visual representations of the delete keys. (These substitute glyphs could also have been specified in the keyboard glyph fields for these items in the menu's 'xmnu' resource.)

The next block is applicable only to the PowerPC target, and only executes if Mac OS 8.5 or later is present. 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).

The block beginning with the call to HMGetHelpMenuHandle appends a menu item with the name "Menus Help" to the Help menu.

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

DoMenuChoice

DoMenuChoice takes the long integer returned by the MenuSelect and MenuEvent calls, extracts the menu ID, the menu item number, and the command ID (if any) and branches according to the menu ID (if no Command ID is present) or the command ID (if present).

Prior to the switch, the menu ID and the menu item number are extracted from the long integer. An immediate return is made 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 the Apple, Font, or Help menus, the relevant application-defined functions are called to further handle the menu item choice. 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 program switches according to the command ID.

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 for checkmarking the Style menu items and changing the current style, and for checkmarking the Size menu items and storing the current size, has been divided between this function and two further-handling functions (doCheckStyleMenuItem and doCheckSizeMenuItem).

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

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