Chapter 13
Menu Manager

Menu Item Styles

The menu manager on system disk 4.0 and prior did not support Outline and shadow text styles as the manual implied. This new version of the menu manager now supports Outlined and Shadowed items. With support of these new styles we have also added the special characters for menu definition:

O	Make the text outlined.
S	Make the text shadowed.

The outline and shadow style fields will also start working with the SetMItemStyle call. To set a style of a menu item, you would pass a style word to the menu manager. A style word is defined by QuickDraw as follows:

Bit0	Bold
Bit1	Italic
Bit2	Underline
Bit3	Outline
Bit4	Shadow

Scrolling Menus

If a menu does not fit on the screen it will scroll. When a menu scrolls, an indicator will appear at the bottom of the menu, as shown in Figure 3. The indicator area itself doesn't highlight, but the menu scrolls as the user drags over it. When the last item is shown, the indicator disappears.

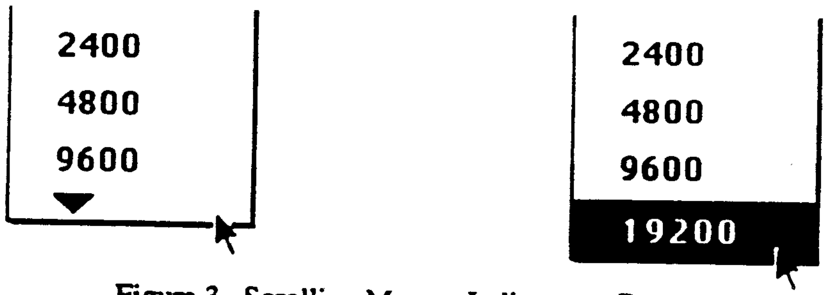

Figure 3. Scrolling Menus: Indicator at Bottom

As soon as the menu starts scrolling, another indicator appears at the top of the menu to show that some items are now hidden in that direction.

If the user drags back up to the top, the menu scrolls back down in the same manner.

The menu scrolls at two speeds depending on where the user drags in the indicator. If the user drags anywhere within the first 5 pixels of the up/down indicators scrolling occurs at a slower rate and dragging anywhere pass this point will allow faster scrolling.

Note: If you have menus in a window and that window is moveable, then dragging the window closer to the bottom of the screen may cause some menus, the next time they are pulled down, to be scrollable. If there are not at least 3 visible items in the menu then the menu will drop below the screen since there must be enough room for at least an up and down arrow and one visible item.

Pop-up Menus

The new menu manager now supports *Pop-up Menus*. These are menus that do not exist in the menu bar, instead they can exist in a window. An example of a window with pop-ups is shown in figure 1.

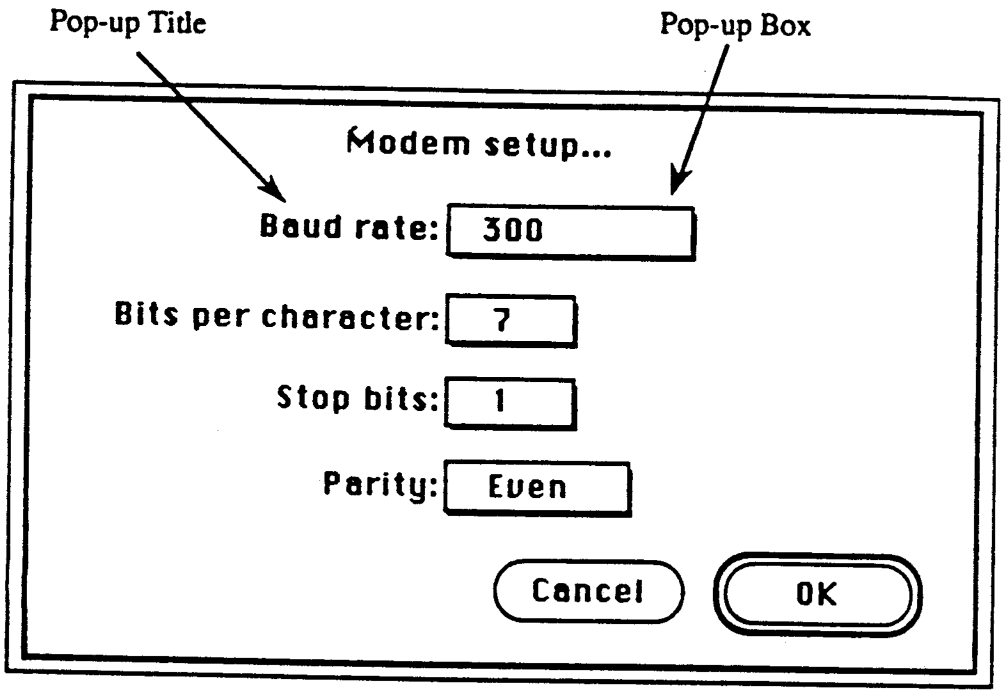

Figure 1. Window With Pop-up Menus

Pop-up menus are used for setting values or choosing from lists of related items. The indication that there is a pop-up menu is a box with a one-pixel thick drop shadow. When the user clicks the mouse in this box, the pop-up menu appears, with the current value highlighted under the pointer, as shown in Figure 2. If the menu has a title, the title is highlighted while the menu is visible.

The pop-up menu acts like other menus: the user can move around in it and choose another item or can move outside it to leave the current value active. If a pop-up menu reaches the top or bottom of the screen (or window if you desire), it will scroll. When scrolling takes place depends on how certain bits are set when the pop-up is created. You can have the pop-up contrained to the window that the pop-up is created in, in which case the pop-up will be scrollable if the menu tries to "pop" out of the window's portRect. You can also have the pop-up appear in the menu manager's port in which case it is contrained by the screen.

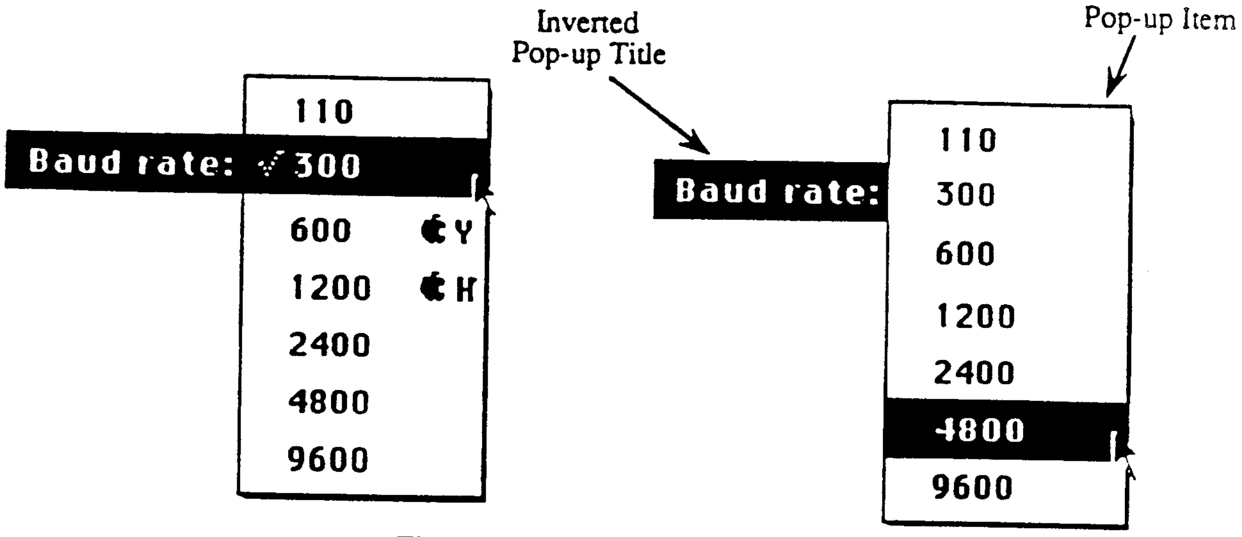

Figure 2. Dragging Through a Pop-up Menu

Type 2 Pop-Ups

There are two types of pop-ups which can be created. Both types are similar in appearance until scrolling has to occur. Figure 2 shows a normal pop-up. The idea behind a "Type2" pop-up is to show as many items in the menu at once as possible. Figure 3 below shows how a pop-up would look if the last item in the Baud Rate pop-up was selected and the pop-up was contrained to the window that its created in. As you can see this limits the visibility of the remaining items in the menu and scrolling is limited to only two items.

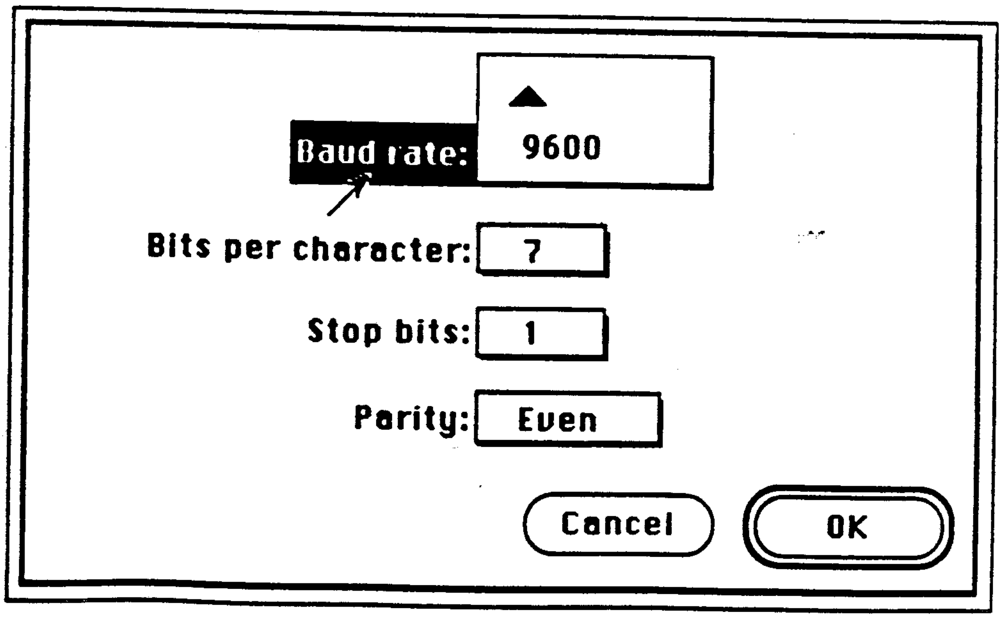

Figure 3.

Figure 4 shows what a type 2 pop-up looks like if it were used in place of the normal pop-up in Figure 3. As you can see, by dragging the mouse over the up arrow indicator the user can eventually scroll all the items in the menu to view. Type 2 pop-ups are very useful when the area that is used to create the pop-up is very limited (i.e. when a pop-up needs to be contrained to the window its in).

When a pop-up needs to be made scrollable, for all the menu items that are not visible, "white space" is created for those items to scroll into. In Figure 4. all the items above 9600 are not visible due to the top boundary contraint. The amount of "white space" created is equal to the number of items that are not visible, or until the bottom boundary contraint is met. This allows the user to see as many items in the menu as possible at one time.within the boundary limitations, without having to select anything.

See PopUpMenuSelect or the Control Manager section on Pop-Up controls on how to create each type of pop-up.

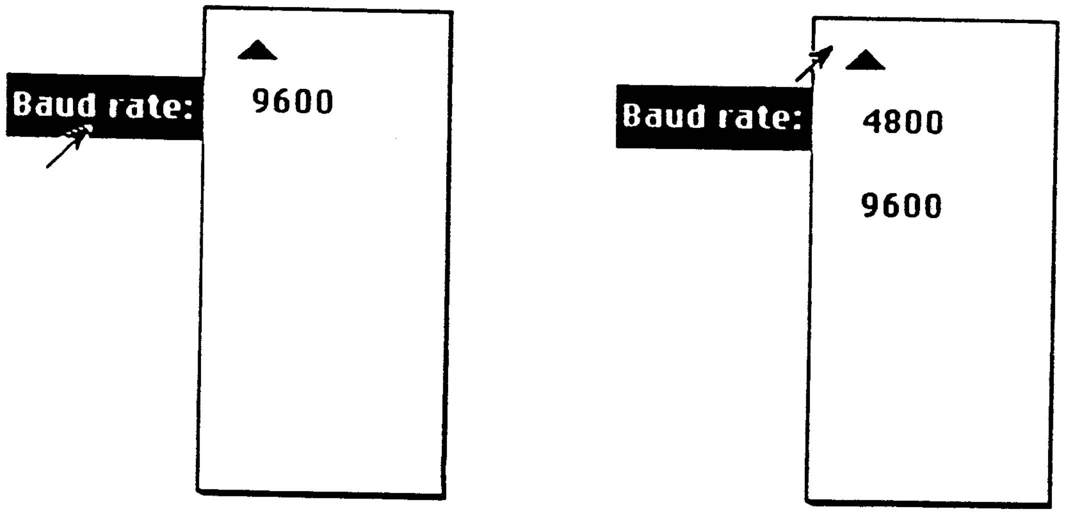

Figure 4.

Pop-up menus will support most of the features and calls that are currently available with standard menus.

- An Apple logo may appear on the right side of the item to show that the item may be invoked from the keyboard followed by a captial letter. If the user presses this key while holding down the Apple key, the menu item is invoked just as if it had been chosen from the menu. *Note*: If a pop-up has the same keyboard equivalent as a regular menu item the pop-up will never receive the keystroke because the regular menu will have handled it first.

- An item can be dimmed to indicate that it is disabled and can't be chosen.

- A mark or icon may appear to the left side of an item to denote the status of an item (i.e. if an item is selected then a check mark may appear next to it) or it may be used to indicate what type of item it may be (i.e. if you had a list of subdirectories, a disk icon may represent the root and a folder icon may represent a subdirectory).

- Each item's text may have its own style.

- If there are more items than can be shown at once inside a menu then the user can scroll the menu. For more details see section on Scrolling Menus.

How To Use Pop-ups

Applications can use pop-up menus two ways.

1. The easiest way is to use NewControl2 to create the pop-up. By making the pop-up a control an application need not worry about drawing the pop-up's box or title, updating of the pop-up after a new selection has been made and can also take advantage of letting TrackControl track the pop-up and also handling command key equivalents if there are any. There are also several bits which can be set to alter the appearance of the pop-up (i.e. don't have title highlight, no title at all, left justify title, etc...). Refer to the Control Manager ERS on how to create a PopUpControl Template to be used as input to NewControl2.

2. The second way to use pop-ups provides a little more flexibility but requires a little more work (This is the way pop-ups are implemented on the Macintosh) The application is responsible for drawing the pop-up box, pop-up title, highlighting of the title, recognizing a mouse-down event in the pop-up box or title, and for changing the entry in the pop-up box after an item has been chosen from the pop-up menu. Similarly, the application is responsible for highlighting the title if the pop-up menu has Command-key equivalents. Once a mouse-down event has been detected in the pop-up box and/or title PopUpMenuSelect is called to bring up the menu and track the mouse. The item ID of the menu item selected is returned. If no item was selected zero is returned. If a menu item was selected then the next time PopUpMenuSelect is called this new selection must be used as input into the call to reflect the latest selection.

Menu Manager Routines for Pop-ups

Each of the following routines operate on the current menu bar. If you are using NewControl2 to create the pop-up you must first set the current menu to the pop-up (i.e. call SetMenuBar using the control handle to the pop-up as input), before you can use any of these calls. If you are using PopUpMenuSelect you must get the handle to the menu you would normally pass to PopUpMenuSelect, insert this menu into the current menu bar calling InsertMenu, make the menu manager call you want, and then remove the menu from the current menu bar by calling DeleteMenu.

List of menu manager calls that should support pop-ups as well:

CalcMenuSize :	Sets menu dimensions, either manually or automatically. Call this if you've just added or deleted an item from the menu so that the menu will be resized correctly.
InsertMItem :	Adds a specified item to the pop-up menu and increments the NumOfItem field in the menu record.
DeleteMItem :	Removes a specified item from the pop-up menu and decrements the NumOfItem field in the menu record. If the item that is being deleted happens to be the currently selected item then the item that shows up in the pop-up rect gets cleared(unless the FDontDrawResult bit is set, in which case no item is shown anyways). The next time the pop-up menu gets selected the first item in the pop-up is the one that lies directly under the pop-up rect, although nothing is selected at this point, i.e. the ctlValue field in the pop-up's control record is zero.
CountMItems :	Returns the number of items, including any dividing lines, in a specified pop-up.
SetMItem :	Replaces the name for a particular pop-up menu item.
GetMItem :	Returns a pointer, handle, or resource ID to the name of an item.
EnableMItem :	Sets a specified pop-up menu item to display normally and allows it to be selected.

DisableMItem :	Sets a specified pop-up menu item to display in dimmed characters and does not allow it to be selected.
CheckMItem :	Sets a specified pop-up menu item to display or to not display a check mark to the left of the item.
SetMItemMark :	Sets a specified pop-up menu item to display or to not display a specified character to the left of an item.
GetMItemMark :	Returns the current character that is displayed to the left of a specified pop-up menu item.
SetMItemStyle :	Sets the text style for a specified pop-up menu item.
GetMItemStyle:	Returns the text style for a specified pop-up menu item.
SetMItemFlag :	Sets a specified item number to be underlined or not underlined and sets the highlighting style.
GetMItemFlag :	Returns the values for a specified item, such as whether it is disabled, underlined, or highlighted.
SetMItemBlink :	Determines how many times all pop-up menu items should blink when selected.
SetMenuBar :	Sets the current menu bar. The control handle to the pop-up is used as input this call.
GetMHandle :	Returns a handle to the pop-ups menu record.
SetMTitleWidth :	Sets the width of the pop-up's title.
GetMTitleWidth :	Gets the width of the pop-up's title.
SetMenuFlag :	Sets the pop-up's menu flag to a specified state. The menuFlag is now a word value. See the expanded description of menuFlag below.
GetMenuFlag :	Returns the pop-up's menu flag. The menuFlag is now a word value. See the expanded description of menuFlag below.
SetMenuTitle :	Sets the pop-up's title to a new name. Input is a pointer to a pascal type string.
GetMenuTitle :	Returns either a pointer, handle or resource ID to the pop-up's title.
SetMenuID :	Specifies a new menu number.

New Data Structures and Resource Support

In order to support resources better, we add a number of new Menu and Menu item definition calls that allow the programmer to specify different kinds of input (pointers, handles and resource IDs). The data structures that support these calls are described below. Use the following data structures for any of the new menu manager calls described below.

Note: Any reference to strings are Pascal type strings.

Note: None of the bits for any of the flags described below have been redefined (so everything is backward compatible). Only those bits that were previously undefined were used for relaying additional information. Also the menuFlag is now a word value instead of a byte value. Previously, in the menu record the byte following the menuFlag was the menuRes flag which was never defined, it is now part of the menuFlag.

MenuItemTemplate

Version	WORD	Must be zero
ItemID	WORD	ID of menu item
ItemChar	BYTE	primary shortcut character
ItemAltChar	BYTE	secondary shortcut character
ItemCheck	WORD	Character to use for checking
ItemFlag	WORD	menu item flags defined below
ItemTitleRef	LONG	Pointer, handle or resource ID of title string

Note: If ItemTitleRef is a pointer then it must point to a PASCAL type string.

ItemFlag

Bit 0	Bold	1 = bold, 0 = not bold
Bit 1	Italic	1 = italic, 0 = not italic
Bit 2	UnderLine	1 = underline, 0 = not underline
Bit 3-4	Reserved	must be zero
Bit 5	XOR	1 = use XOR to highlight selection, 0 = don't use XOR
Bit 6	Divider	1 = Draw Divider bar, 0 = No devider bar
Bit 7	Disabled	1 = disabled, 0 = enabled
Bit 8	Reserved	must be 0
Bit 9	Reserved	must be 0
Bit 10	Reserved	must be 0
Bit 11	Outline	1 = outline, 0 = not outline
Bit 12	Shadow	1 = shadow, 0 = not shadow
Bit 13	Reserved	must be zero
Bit 14-15	TitleRefType	0 for Pointer, 1 for handle, 2 for resource ID

MenuTemplate

Version	WORD	Must be zero
MenuID	WORD	ID of menu
MenuFlag	WORD	menu flags defined below
MenuTitleRef	LONG	Pointer, handle or resource ID of title string
Item1Ref	LONG	Pointer, handle or resource ID of menu item 1
Item2Ref	LONG	for item 2
...		
ItemNRef	LONG	for item N
Terminator	LONG	Must be zero

Note: If MenuTitleRef is a pointer, then the pointer must point to a PASCAL type string.

MenuFlag

Bit 0-2	Reserved	must be zero
Bit 3	AllowCache	1 = Cache Menu, 0 = Don't cache menu
Bit 4	Custom	1 = Menu is custom, 0 = menu is standard.
Bit 5	XOR	1 = use XOR to highlight selection, 0 = don't use XOR
Bit 6	Reserved	must be 0
Bit 7	Disabled	1 = disabled, 0 = enabled
Bit 8-11	Reserved	must be 0
Bit 12-13	ItemRefType	0 for Pointer, 1 for handle, 2 for resource ID
Bit 14-15	TitleRefType	0 for Pointer, 1 for handle, 2 for resource ID

Note: All the ItemRefs for a particular menu must be of the same ref type, all must be a pointer, handle or resource ID.

MenuBarTemplate

Version	WORD	Must be zero.
MenuBarFlag	WORD	menu bar flag defined below
Menu1Ref	LONG	Pointer, handle or resource ID of Menu 1
Menu2Ref	LONG	for Menu 2
...		
Menu3Ref	LONG	for Menu 3
Terminator	LONG	Must be zero

MenuBarFlag

Bit 0-13	Reserved	must be zero
Bit 14-15	MenuRefType	0 for Pointer, 1 for handle, 2 for resource ID

The new calls that provide support for these data types take an additional input that describes what the reference is. The three possible values are:

RefIsPointer	= 0
RefIsHandle	= 1
RefIsResource	= 2

New Calls

PopUpMenuSelect Call $3C0F

Input

ItemIDSpace	: WORD	Space for id of selected item.
Selection	: WORD	Item ID of current selection
CurrentLeft	: WORD	Left side of popped-up menu
CurrentTop	: WORD	Top of current selection in popped-up menu
Flag	: WORD	Currently only bit 6 of flag is defined.
MenuHandle	: LONG	Handle to a Menu

Output

ItemID	: WORD	id of item selected

Works the same as MenuSelect except that it tracks a pop-up menu instead. MenuHandle is the handle returned from NewMenu or NewMenu2. ItemID equals zero if no item in the pop-up is selected. If bit 6 of the flag is set then popupmenuselect will draw the pop-up with white space, i.e. if there is not enough room in the grafport to draw the entire pop-up then white space will be added until the entire pop-up fits or until we overstep the portrect's boundaries. See Figure 5.
Note: The coordinates CurrentLeft and CurrentTop must be in global coordinates. The programmer must also be aware that the pop-up menu is drawn in whatever port is set to at the time PopUpMenuSelect is called. For example if the programmer wants the pop-up menu to appear anywhere on the screen (i.e. the menu can "pop" out of a window) then the port should be set to the menu manager's port before calling PopUpMenuSelect. The pop-up menu is contrained by the intersection between the port rectangle, the visible region and the clip region. By altering these values you can alter how the pop-up should be contrained.

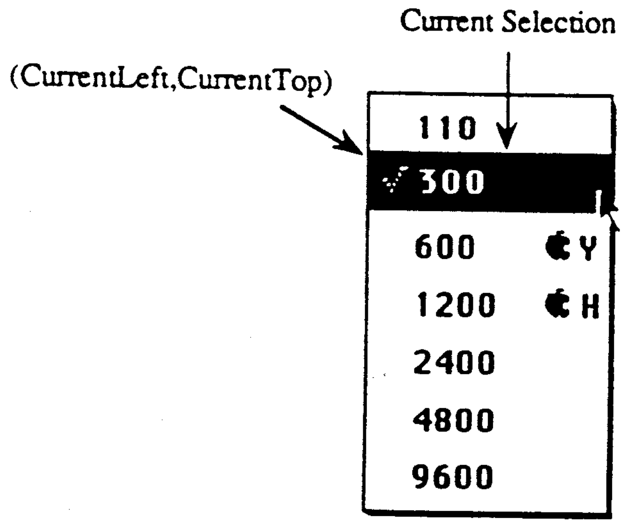

Figure 5.

GetPopUpDefProc Call $3B0F

Input

Space	: LONG	

Output

DefProcPtr	: LONG	Pointer to defProc handle

Returns a pointer to the control defProc that implements pop-up menus. This call is intended primarily for the use of the control manager.

DrawPopUp Call $3D0F

Input

Selection	: WORD	; item number to be selected
Flag	: WORD	;
Right	: WORD	; right edge of pop up rect
Bottom	: WORD	; bottom of pop up rect
Left	: WORD	; left of pop up rect
Top	: WORD	; top of pop up rect
MenuHandle	: WORD	; handle to menu being drawn

Output

none

This is an internal MenuManager routine used to draw a pop up menu. It would not normally be called by an application.

NewMenuBar2 Call $430F

Input

Space	: LONG	space for resulting handle
RefDescriptor	: WORD	describes what next parameter is
MenuBarTemplateRef	: LONG	pointer, handle or resource id of MenuBarTemplate.
theWindowPtr	: LONG	Pointer to window's port or NIL for system menu bar.

Output

ResultHandle	: LONG	Menu Bar Handle

This call creates an entire menu bar at once using a menu bar template as input. The RefDescriptor tells the call how the temple is referenced.

NewMenu2 Call $3E0F

Input

Space	: LONG	space for resulting handle
RefDescriptor	: WORD	describes what next parameter is
MenuTemplateRef	: LONG	pointer, handle or resource id of MenuTemplate.

Output

ResultHandle	: LONG	Menu Handle

Works the same as NewMenu except that RefDescriptor and MenuTemplateRef are passed instead of pointer to Menu definition data.

InsertMItem2 Call $3F0F

Input

RefDescriptor	: WORD	describes what next parameter is
MenuItemTemplateRef	: LONG	pointer, handle, or resource ID of MenuItemTemplate.
InsertAfter	: WORD	ID of item after which this is to be inserted. 0 to add to front, $FFFF at end.
MenuNum	: WORD	ID of menu that this will be in.

Output
none

Works the same as InsertMItem except that RefDescriptor and MenuItemTemplateRef are passed instead of pointer to menu item definition data.

SetMenuTitle2 Call $400F

Input

RefDescriptor	: WORD	describes what next parameter is
TitleRef	: LONG	pointer handle or resource ID of title string.
MenuNum	: WORD	ID of menu that gets new title.

Output
none

Works the same as SetMenuTitle except that RefDescriptor and TitleRef are passed instead of pointer to menu title.

SetMItem2 Call $410F

Input

RefDescriptor	: WORD	describes what next parameter is
MenuItemTemplateRef	: LONG	pointer, handle or resource ID of menu item template
MenuItem	: WORD	ID of item that will be changed.

Output
none

Works the same as SetMItem except that RefDescriptor and MenuItemTemplateRef are passed instead of pointer to menu item definition data.

SetMItemName2 **Call $420F**

Input

RefDescriptor	: WORD	describes what next parameter is
TitleRef	: LONG	pointer, handle or resource ID of title string
MenuItem	: WORD	ID of item that will get new title

Output

none

Works the same as SetMItemName except that RefDescriptor and TitleRef are passed instead of pointer to menu item string.

HideMenuBar **Call $450F**

Input

none

Output

none

HideMenuBar takes the system menu bar, if there is one and hides it. It hides it by taking the menu bar's rectangle and adding it back to the desktop region. It sets the menu bar's invisible bit, resets the 2 thru 9 scan lines which were changed so that the apple logo's colors would appear correctly and then refreshes the desktop. Note: all subsequent calls to _DrawMenuBar or _FlashMenuBar are ignored since the menu is considered invisible, and cannot be made visible until ShowMenuBar is called.

ShowMenuBar **Call $460F**

Input

none

Output

none

ShowMenuBar takes the system menu bar, if there is one and makes it visible. The system menu bar is made visible by subtracting the menu bar's rect from the desktop region. The invisible bit in the menu bar record is cleared, the SCBs in the menu bar are reinitialized for the apple logo, the desktop is refreshed and finally the menu bar is redrawn.

Menu Manager Updates for the ToolBox Reference Manual.

Correction: *SetBarColors* does not work as anticipated if there is more than one menu bar being used. When a menu bar is created, it is created using a default color table. SetBarColors changes the entries in this default color table not the actual pointer to the table itself. Therefore any menu bars that have already been created, or will be created will now use these new colors set by SetBarColors. If you are using more than one menu bar and you want to change the default color table then the only way to do this without affecting any other menu bars is to actually construct the new color table yourself and then to replace the pointer in the menu bar record's ctlColor field with the pointer to the desired color table.

Correction: *SetSysBar* does not make the system bar the current menu bar.

You can now have empty menus.
Creating an empty menu:

```
        dc.b    '$$ Empty Menu \N1',$00    ; menu's title and ID
        dc.b    $00                        ; first character in first item line is either a null or return
                                           ; byte, this signifies the end of the menu definition.
```

Change History

31 Jan 89 Harry Yee

Added section "Menu Manager Updates for the ToolBox Reference Manual". Added additional info on calls SetBarColors and SetSysBar into this section.

Added information about Type2 pop-ups. Added more info. on PopUpMenuSelect. Added description of new calls HideMenuBar and ShowMenuBar. Added info. in DeleteMItem when deleting the current selection. The template section now states that any strings that are pointers are pointers to PASCAL type strings.