Cortland Menu Manager

Dan Oliver

02/18/86 Initial release

05/08/86 Many parameter changes to accommodate menu speedups. Menu record changes and an alternative method of defining menus, see MENU STRINGS. Additional Mac type calls to help portability. Many unnecessary features that slowed things down have gone away. Someone, I'm not sure who, suggested the name 'Fall Down' menus instead of 'Hot' menus. I think 'fall down' is more precise and relates better to 'pull down' menus, so it has now replaced the term 'hot'. New calls CheckItem, SetItemMark, GetItemMark, EnableItem, DisableItem, NewMenu, DisposeMenu, SetMenuID, SetItemID, SetSysBar, GetSysBar, and InitPalette.

This ERS describes the Menu Manager, the part of the Cortland Toolbox that allows you to create sets of menus, and allows the user to choose from the commands in those menus.

You should already be familiar with the Cortland Toolbox Event Manager.

ABOUT THE MENU MANAGER

The Menu Manager supports the use of menus which can be part of the Cortland user interface. Menus allow users to examine all choices available to them at any time without being forced to choose one of them, and without having to remember command words or special keys. The Cortland user simply positions the cursor in the menu bar and presses the mouse button over a menu title. The application then calls the Menu Manager, which highlights selected title (by redrawing it with its InvertColor) and "pulls down" the menu below it. As long as the mouse button is held down, the menu is displayed. Dragging through the menu causes each of the menu items (commands) in it to be highlighted in turn. If the mouse button is released over an item, that item is "chosen". The item blinks briefly to confirm the choice, and the menu disappears.

When the user chooses an item, the Menu Manager tells the application which item was chosen, and the application performs the corresponding action. When the application completes the action, it removes the highlighting from the menu title, indicating to the user that the operation is complete.

If the user moves the cursor out of the menu with the mouse button held down, the menu remains visible, though no menu items are highlighted. If the mouse button is released outside the menu, no choice is made: The menu just disappears and the application takes no action. The user can always look at a menu without causing any changes in the document or on the screen.

MENU BAR

A menu bar is an outlined rectangle that holds the titles of all the menus associated with the bar. A menu may be enabled or temporarily disabled. A disabled menu can still be pulled down, but its title and all the items in it are dimmed and not selectable.

Keep in mind that if your program is likely to be translated into other languages, the menu titles may take up more space. If you're having trouble fitting your menus into the menu bar, you should review your menu organization and menu titles.

THE SYSTEM MENU BAR

There can be one special type of menu bar which is called the System Menu Bar. There can only be one system menu bar on the screen at one time. The system menu bar always appears at the top of the Cortland screen; nothing but the cursor ever appears in front of it. In applications that support desk accessories, the first menu should be the desk accessory menu (the menu whose title is a colored apple symbol). The desk accessory menu contains the names of all available desk accessories. When the user chooses a desk accessory from the menu, the title of a menu belonging to the desk accessory may appear in the menu bar, for as long as the accessory is active, or the entire menu bar may be replaced by menus belonging to the desk accessory.

Color number 1 is reserved for drawing the Apple logo as the title for the desk accessory menu. Therefore, color number 1 should not be used as the BarColor, InvertColor, or the Outline color. The color can be used for menus, items, nonsystem menu bars, and the rest of the screen.

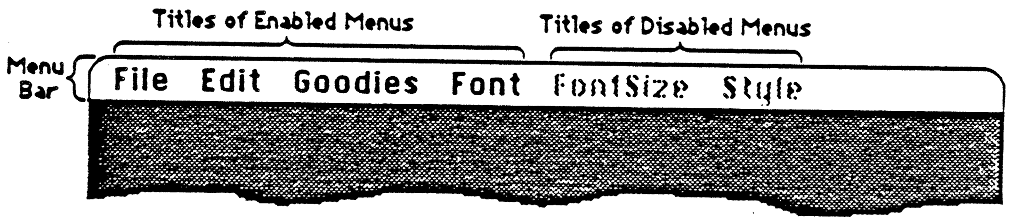
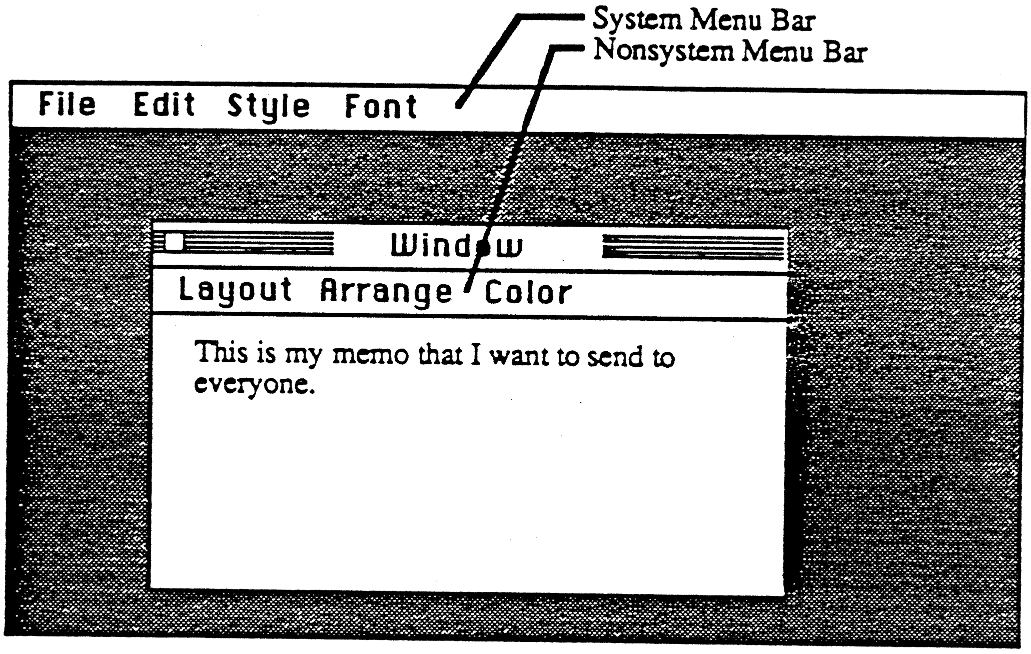


Figure 1. The System Menu Bar

NONSYSTEM MENU BARS

In addition to the System Menu Bar your application can have various nonsystem menu bars. These can appear any where on the screen and in windows. Nonsystem menu bars are provided to give you more flexibility and to address the limited resolution in 320 mode. Nonsystem menu bars should be used moderately, if at all. Nonsystem menu bars perform in the same manner as the System Menu Bar, although the fall down feature (discussed later) should generally not be used for nonsystem menu bars.

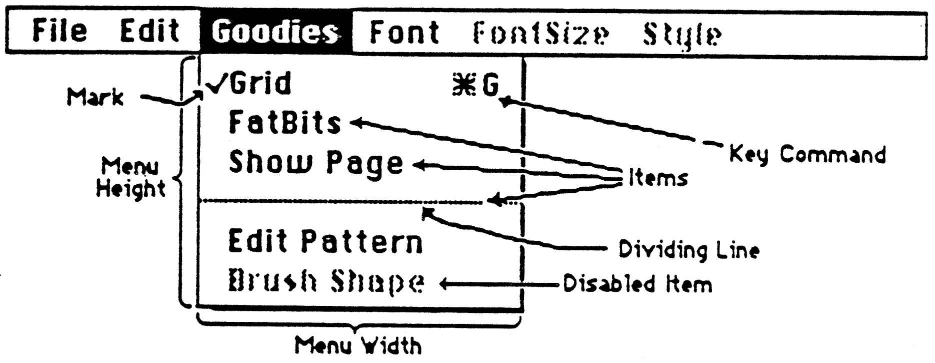


This is my memo that I want to send to everyone.

APPEARANCE OF MENUS

A standard menu consists of a number of menu items listed vertically inside a shadowed rectangle. A menu item may be the text of a command, a solid color, or just a line dividing groups of choices (see Figure 2). Menus always appear in front of everything else, except the cursor. The menu in figure 2 is a menu with 6 items including one dividing line.

Figure 2. A Standard Menu.



Each item can have a few visual variations from the standard appearance:

- A mark may appear on the left side of the item, to denote the status of the item or of the mode it controls. See SetItemMark, GetItemMark, and CheckItem.

- A apple symbol on the right side of the item, to show that the item may be invoked from the keyboard (that is, it has a keyboard equivalent), followed by a capital letter. Pressing this key while holding down the Command key invokes the item just as if it had been chosen from the menu. See MenuKey .

- Each item's text may have its text style. See SetItemStyle and GetItemStyle.

- A dimmed appearance, to indicate that the item is disabled, and can't be chosen (dividing lines should always be disabled). See DisableItem and EnableItem.

- Each item may be just a solid color.

- Any menu may be drawn directly by the application and might contain anything (see DEFINING YOUR OWN MENUS).

Note: If an item without text is dimmed it will appear as the menu background (I'm in need of something better).

If the standard menu doesn't suit your needs—for example, if you want more graphics, or perhaps a nonlinear text arrangement—you can define a custom menu that, although visibly different to the user, responds to your application's Menu Manager calls just like a standard menu (see DEFINING YOUR OWN MENUS).

KEYBOARD EQUIVALENTS FOR COMMANDS

Your program can set up a keyboard equivalent for any of its menu commands so the command can be invoked from the keyboard with the Command key (apple key). The character you specify for a keyboard equivalent will usually be a letter. The user can type the letter in either uppercase or lowercase. For example, typing either "C" or "c" while holding down the Command key invokes the command whose equivalent is "C".

> **Note:** For consistency between applications, you should specify the letter in uppercase in the menu.

USING THE MENU MANAGER

To use the Menu Manager, you must have previously initialized QuickDraw. For user interaction you must use the Event Manager.

The first Menu Manager routine to call is the initialization procedure, **InitMenus**.

Your program then must define the menu bar, menus, and items by providing a Menu String to **NewMenu**, see MENU STRING. Your application can set up a system menu bar by using **SetMenuBar** to replace the system menu bar, or **InsertMenu** to add yours to the default system menu bar. **FixMenuBar** may be of use in setting default sizes.

After created, **DrawMenuBar** will put the menu on the screen.

To handle user input your application calls **MenuSelect**, **MenuKey**, or **CheckFallDown**.

MenuSelect should be called with the system menu bar when the Window Manager's **FindWindow** function returns an in System Menu Bar value after your application receives a mouse-down event. Or, MenuSelect should be called with a pointer to any application menu bars your may have when you detect a mouse-down event over them. **MenuSelect** will pull down the appropriate menu, and retain control—tracking the mouse, highlighting menu items, and pulling down other menus—until the user releases the mouse button.

When your application receives a key-down event with the Command key held down, it should call the **MenuKey** function, supplying it with the character that was typed and the menu bar to check. Applications should respond the same way to auto-key events as to key-down events when the Command key is held down if the command being invoked is repeatable.

CheckFallDown should be called if you are using a fall down system menu bar. It should be called everytime there is a null event. If **CheckFallDown** finds the cursor over the fall down area of a menu title it will highlight the title and pull down the menu. It will then track the mouse, highlighting any items it moves over and unhighlighting any it leaves. **CheckFallDown** will return a selection if the user presses and releases the mouse button over an enabled item. The selection is the item the cursor is over when the button is released, so the user can drag the cursor before the selection is made. **CheckFallDown** will also return if the cursor leaves the menu plus an invisible border around the menu.

MenuSelect, **MenuKey**, and **CheckFallDown** will all return a WORD reporting what action the user performed.

- If the WORD returned is 0, the application should just continue to poll for further events.

- If the WORD is nonzero, the high BYTE will be a menu ID and the low BYTE an item ID of the menu and item selected by the user. Your application should then envoke an action which is specific to the selected item. Only after the action is completely finished (after all dialogs, alerts, or screen actions have been taken care of) should the application remove the highlighting from the menu bar by calling **HiliteMenu**, signaling the completion of the action.

Note: The Menu Manager will try to automatically save and restore the screen behind the menu, or tell the Window Manager to update the screen. However, if you are not using the Window Manager and the Menu Manager can not allocate a buffer large enough to save the screen behind the menu, your application will have to update the screen area after a menu has been pulled down. See **CheckRedraw**.

If your menu bar, or items in a menu, are going to change while on the screen you can use **SetMenuTitle**, **InsertMenu**, **DeleteMenu**, **SetItem**, **InsertItem**, and **DeleteItem** to rearrange the MenuList and the ItemList.

There are several miscellaneous Menu Manager routines that you normally won't need to use. **CalcMenuSize** calculates the dimensions of a menu and is called by **FixMenuBar**. **CountMItems** counts the number of items in a menu. **GetMPtr** returns a pointer to the menu in the menu list. **FlashMenuBar** inverts the menu bar, or just a menu title. **SetItemBlink** controls the number of times a menu item blinks when it's chosen.

MENU STRINGS

Menus may be created by passing a text string to **NewMenu** which will parse the string, allocate enough memory for necessary records, and initialize those records. The menu string can be edited using a word processor, thus allowing users to easily customize their own menus. An example of a menu string is:

```
>Title 1
-Item string 1
-Item string 2
-Item string 3
>Title 2
-Item string 1
-Item string 2
.
```

This is a simple menu list of two menus, the first with 3 items, and the second with 2 items. The first character on a line denotes the start of a menu or an item in a menu. Lines are separated by returns. The character to denote a title is whatever the very first character is. The character to denote items is the first character on subsequent lines that is different from the title character. And lastly, a third character, different from the menu and item character, denotes the end of the menu string.

If you would like to get fancier, you can add some special characters.

\	Beginning of special characters.
*	Followed by a character to be used as a keyboard equivalent.
C	Followed by a character to be used to mark the item.
B	Bold the text.
I	Italize the text.
U	Underline the text.
V	Places a dividing line under the item without using a separate item.
D	To dim (disable).
X	Use special XOR highlighting.

All the special keys pertain to items, but only \, D, and X pertain to menu titles. The keys could be used in the following manner:

```
>Title 1
-Item string 1
-Item string 2\BC√*RXDU
-Item string 3\I
>Title 2\X
-Item string 1\X*BB
-Item string 2\X*UU
.
```

Where:

	Keyboard equivalent	Checked	Bolded	Italicized	Underlined	Diving line	Disabled	XOR highlighting
>Title 1								
-Item string 1								
-Item string 2\BC√*RXDU	R	√	√		√		√	√
>Title 2\X								√
-Item string 3\I				√				
-Item string 1\X*BB	B		√					√
-Item string 2\X*UU	U				√			√

Some more special stuff. Using just the @ symbol in a title will give you the Apple logo.

Note: Don't use the X special key (XOR highlighting) with the Apple logo.

Note: To get the Apple logo the @ must follow the character denoting a menu title, and then be followed by an end of line mark (return). Do not place a space before or after the @ like you might with other menu titles.

Do not press the Open Apple key when entering a key for keyboard equivalents.

There is no way to include a '\' in a text string. It will always be seen as the beginning of special characters.

Color Menus

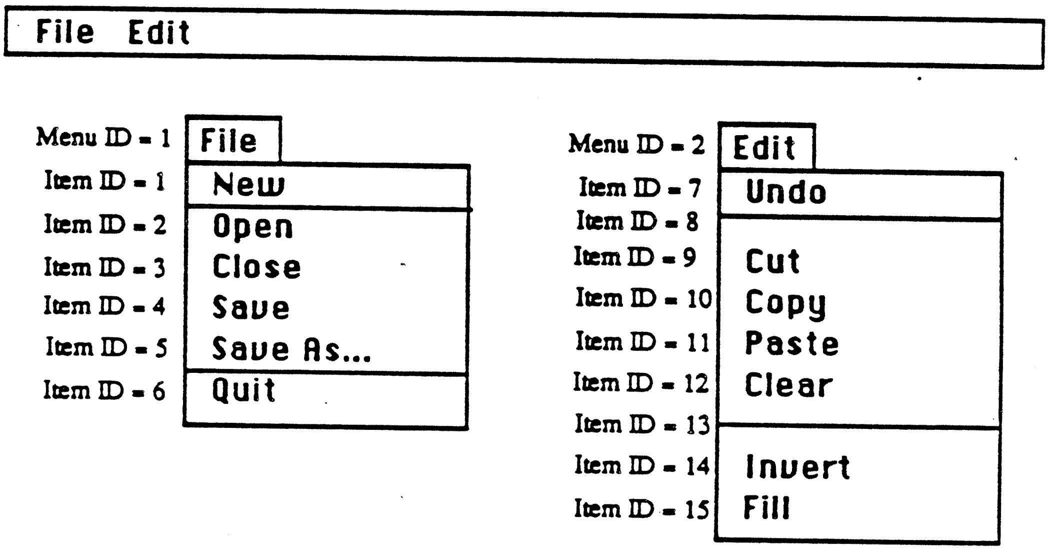
You can create a color menu by using a single lower case letter, a-p, as an item's text string. Lower case letter 'a' would be color number 0, letter 'p' is color number 15. However, you can not mix color items with text items. So, if the first item is a single lower case letter the menu will be considered a color menu. Otherwise the menu will be an all text menu.

A color item can be disabled, checked, and have a keyboard equivalent.

ID Number Assignment

ID numbers are assigned in the order of appearance. The first menu has an ID of 1, as does the first item. After that, the menu ID is incremented and assigned to the next menu found, and the same for items.

The next example shows a menu bar with two menus and the ID numbers that would be assigned to the menus and items.



ID numbers could be changed to anything you would like after the NewMenu call, see SetItemID, GetItemID, SetMenuID, and GetMenuID.

MENU RECORDS

Direct access to records should be limited to custom menus.

The Menu Manager uses three different kinds of records; MENUBAR, MENU, and ITEM. A MENUBAR contains a MenuList, a linked list of pointers to MENUs, and a MENU contains a ItemList, a linked list of pointers to ITEMs. Each record also has data telling how each object is drawn. See CREATING A MENU IN YOUR PROGRAM to see how each parameter relates to what the menu does.

MENUBAR record:

NextCtrl	LONG	Pointer to next control (see Control Manager).
CtrlType	BYTE	Control type, 1 = menu bar.
Bar	RECT	Rectangle of the menu bar.
FallDown	BYTE	Number of pixels of fall down area, zero = no fall down area.
BarColor	BYTE	Color of menu bar and text.
InvertColor	BYTE	Color of menu bar and text when inverted.
Outline	BYTE	Color of outlines and underlines.
BarFlag	BYTE	See figure 3.
MenuList	LONG	Pointer to first MENU in menu bar, or zero if no menus.

MENU record:

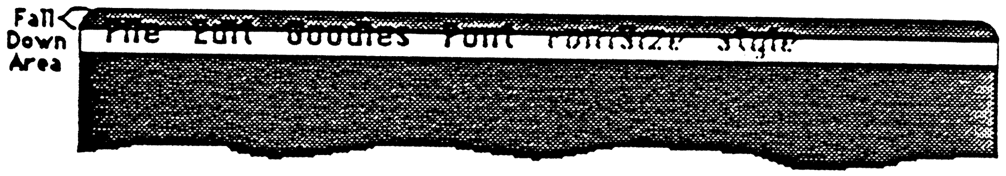
NextMenu	LONG	Pointer to next menu in menu bar, zero if this is the last.
MenuID	WORD	Set by application, returned when a selection is made.
MenuWidth	WORD	Width of menu.
MenuHeight	WORD	Height of menu.
MenuProc	LONG	Address of menu definition procedure, zero for noncustom.
TitleWidth	WORD	Width of title in menu bar.
TitleName	LONG	Pointer to title's string, where the first byte is the length.
MenuFlag	BYTE	Enabled, normal, xor, and type flags (see figure 4).
ItemList	LONG	Pointer to first ITEM in menu, or zero if no items.

ITEM record:

NextItem	LONG	Pointer to next item in menu, zero if this is the last.
ItemID	WORD	Item's ID number, selected by application (non zero).
ItemName	LONG	Either a pointer to string or color value.
ItemChar	BYTE	Key board equivalent.
ItemCheck	BYTE	Character to used to mark an item, zero = no mark.
ItemFlag	BYTE	Enable, underline, xor, text style (see figure 5).

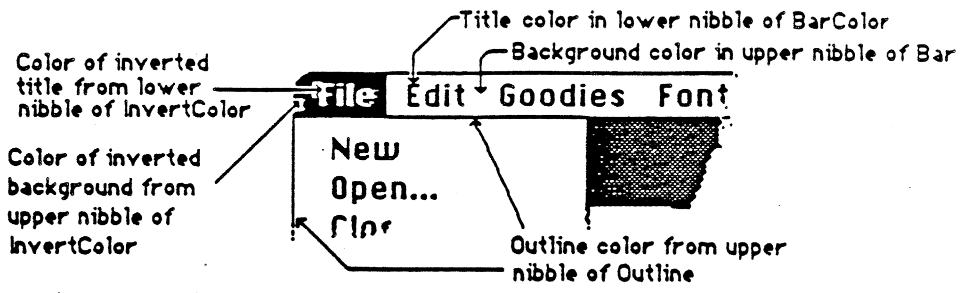
MENU BAR RECORD

NextCtrl - LONG	Pointer to next control in list, zero means this is the last control. Can be set to zero if you are not using the Control Manager.
CtrlType - BYTE	Control type, 1 = menu bar. See Control Manager for other types of controls. Can be set to zero if you are not using the Control Manager.
Bar - RECT	TopSide, LeftSide, BottomSide, RightSide. This bar will have a solid background with the color in BarColor if normal, and InvertColor if inverted. The invert bit is found in BarFlag. The rectangle will be outlined with a solid line of the color in Outline. Menu bars all have square corners except the system menu bar which has round corners in the upper left and upper right corners.
FallDown - BYTE	Number of pixels in fall down area. Used by **CheckFallDown** to see if the user has moved the cursor into the fall down area of the menu. The fall down area is the TopSide, LeftSide, and RightSide in Bar, and the TopSide plus fall down for the bottom side. If fall down equals zero a menu can not be activated in this way.



Possible fall down Area

BarColor - BYTE	The high-order NIBBLE is the background color of the menu bar when not inverted, and menu background color. The low-order NIBBLE is the color of menu titles when not inverted, and the color of item text. Color number 1 should not be used if the menu bar is to be a system menu bar.
InvertColor - BYTE	The high-order NIBBLE is the background color of the menu bar when inverted, and item background color when highlighted. The low-order NIBBLE is the color of menu titles when inverted, and the color of item text when highlighted. Color number 1 should not be used if the menu bar is to be a system menu bar.
Outline - BYTE	Low-order NIBBLE is the color used to draw the outine for the menu bar, menu, and any item dividers. Color number 1 should not be used if the menu bar is to be a system menu bar.

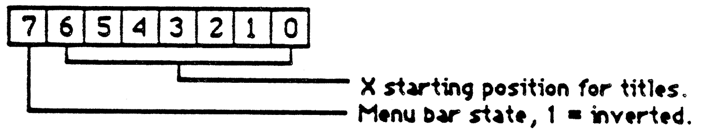

Title color in lower nibble of BarColor
Background color in upper nibble of Bar
Color of inverted title from lower nibble of InvertColor
Color of inverted background from upper nibble of InvertColor
File
Edit
Goodies
Font
New
Open...
Outline color from upper nibble of Outline

BarFlag - BYTE

Bit 7, the high-order bit, is the current state of the menu bar, 1 means the bar should be drawn as inverted. FlashMenuBar flips this bit every time it is called. To start with, this bit should be 0.

Bits 6-0 is the number of pixels from the left side of the menu bar to the left side of the first menu title. The maximum number is 127. For menu bars with square corners a 1 should be used to get the titles as far to the left as possible. For menu bars with round corners at least a 10 should be used to keep the inverted title inside the menu bar. A zero will write over the menu bar's left outline.

Figure 3. MENUBAR.BarFlag


7 6 5 4 3 2 1 0
X starting position for titles.
Menu bar state, 1 = inverted.

MenuList - LONG

Pointer to first menu, left most in the menu bar. After this pointer each menu points to the next menu on the menu bar.

MENU RECORD

NextMenu	LONG	Pointer to next menu in menu bar, or zero if this is the last menu in the menu bar. The pointer points to the menu to the right of its self.
TitleWidth	WORD	Width of a title area. This is the selectable area for a title and is the area inverted when a title is highlighted. Title height is the height of the menu bar.
MenuID	WORD	Any value you would like. This value is returned when a selection is made. Value should not be zero. One possibility is to store the low address of the handling routine here. That way it can be pushed and RTSed to.
MenuWidth	WORD	Width of the menu. This value can be computed and set by calling **CalcMenuSize**.
MenuHeight	WORD	Height of the menu. This value can be computed and set by calling **CalcMenuSize**.
MenuProc	LONG	Address of routine that handles menu draw, zero for standard.
TitleName	LONG	Pointer to string to use as menu's title. It will be drawn using the system with the colors found in the MenuBar record: Normal: Background from high order nibble BarColor. Text color from low order nibble of BarColor. Selected: Background from high order nibble InvertColor. Text color from low order nibble of InvertColor. The first BYTE of the string has to be the length of the string followed by ASCII characters.

MenuFlag	BYTE	Bit 7 can be set to disable a menu. If set, the menu's title will appear dimmed but still selectable, and all the menu's items will be dimmed and not selectable.

Bit 6 is set to invert the menu's title when drawn, if it is clear the title will be drawn normally.

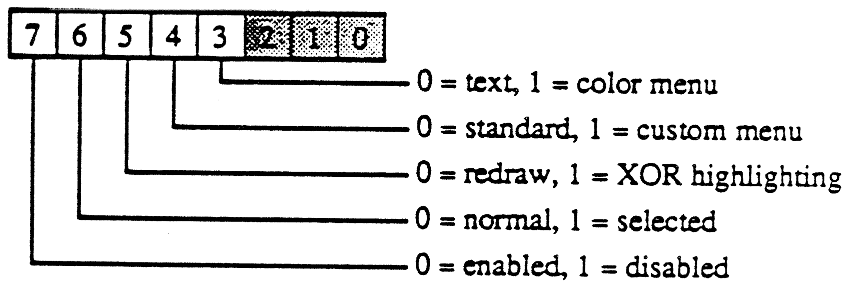
Bit 5 is set to use special XOR mode highlighting. When set the title area will be inverted. For example: black text on white would become white text on black. However, some color combinations may yield undesirable effects. If bit 5 is set InvertColor will not be used for drawing the title. If bit 5 is clear the title and its background will be redrawn using InvertColor when selected.

> **Note:** You should not use XOR with the Apple logo on the system menu bar.

Bit 4 is set if you want to draw and track your own menu. See DEFINING YOUR OWN MENUS.

If bit 4 is clear the Menu Manager will draw and track the menu for you. There are two standard menus. If bit 3 is clear, the menu is all text. If bit 3 is set the menu is all color items. See SPECIAL EFFECTS.

Figure 4. MENU.MenuFlag


7 6 5 4 3 2 1 0
0 = text, 1 = color menu
0 = standard, 1 = custom menu
0 = redraw, 1 = XOR highlighting
0 = normal, 1 = selected
0 = enabled, 1 = disabled

ItemList	LONG	Pointer to the first item, top most, in the menu, or zero if there are no items in the menu. After this pointer, each item points to the item below it in the menu. The item number returned by MenuSelect and MenuKey is how far down this list the item is.

ITEM RECORD

NextItem LONG	Pointer to the next item in the menu, or zero if this is the last item in the menu. This pointer will point to the item appearing below this one in the menu.
ItemID - WORD	Any value you would like. This value is returned when a selection is made. Value should not be zero, and is of no matter if the item is always disable, like a divding line. One possibility is to store the low address of the handling routine here. That way it can be pushed and RTSed to.
ItemName - LONG	For text menus: Pointer to the item's string. For color menus: The low WORD is the item's color.
ItemChar - BYTE	Character to use as keyboard equivalent, zero for none. See KEYBOARD EQUIVALENTS.
ItemCheck - BYTE	Character to mark item. Appears to the left of the item. Zero for no mark.

ItemFlag - WORD

Bit 7 is clear if the item is enabled (selectable), it is set if the item is disabled (dimmed).

Bit 6 should be set to use an underline as a dividing line. See SPECIAL EFFECTS for more information about underlines.

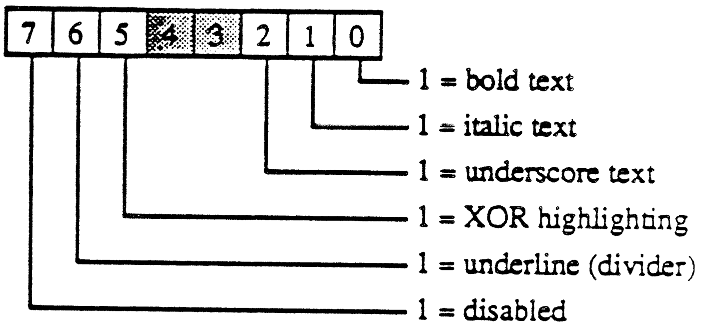
Bit 5 is set to use special XOR mode highlighting. When set, the item area will be inverted. For example: black text on white would become white text on black. However, some color combinations may yield undesirable effects. If bit 5 is set, InvertColor will not be used for drawing the item. If bit 5 is clear, the item and its background will be redrawn using InvertColor when selected. XOR should be used whenever possible.

Bit 2 is set to underscore the item's text. This is not the same thing as underline as a divider.

Bit 1 is set to italicize the item's text.

Bit 0 is set to bold the item's text.

Figure 5. ITEM.ItemFlag



DEFINING YOUR OWN MENUS

You can create your own custom menu if you'd like by writing a menu definition procedure and storing its address in MENU.MenuProc.

The Menu Manager will call your routine when something involving your menu needs to happen. You will have the opportunity to perform the action yourself, or tell the Menu Manager to go ahead and complete the operation.

Input to your routine will be:

	PUSH:LONG	ReturnFlag - space for return flag.
;		
	PUSH:WORD	TaskNum - task number to perform.
	PUSH:LONG	Parameter - parameter.
	PUSH:LONG	BarPtr - pointer to menu bar.
	PUSH:LONG	MenuPtr - pointer to menu.

The tasks you will be asked to perform:

Highlight the menu's title.

Task	= $0001
Parameter	= $00000000 to draw title normal,
	$00000001 to draw title selected.
ReturnFlag	= $00000000 for Menu Manager to perform task.

Draw the menu. The screen will have already been saved and the menu framed.

Task	= $0002
Parameter	= pointer to menu's RECT.
ReturnFlag	= $00000000 for Menu Manager to perform task.

Tracking the user.

Task	= $0003
Parameter	= POINT of cursor, low WORD is Y, high WORD is X.
ReturnFlag	= $00000000 to continue tracking, else return to caller with
	ReturnFlag = Menu ID in high WORD, ItemID in the low.

Selection made. The screen will be restored for you after you complete this task.

Task	= $0004
Parameter	= POINT of selection, low WORD is Y, high WORD is X.
ReturnFlag	= Menu ID in high WORD, ItemID in the low WORD.

SPECIAL EFFECTS

Here are a couple of different ways to approach functions already covered.

Fall Down Menus

The standard way to allow the user to make menu selections is to move the cursor over the menu's title, press the mouse button, drag down over the items, and release the button over the desired item.

Another way is to allow the user to make menu selections by moving the cursor into a 'fall down' area on the menu bar, this would cause the menu to come down without the user pressing the button. Now the user can move the cursor down over the items, press and release the mouse button over an item to select it. To leave the menu without making a selection the user would move the cursor out of the menu and the menu would go away. Actually there is an invisible border around the menu that would have to be crossed before the menu went away.

Fall down menus have a limited use. They should only be used in the system menu bar, and then only as an option. A control in the Control Panel could be used to select the height of the fall down area on the system menu bar so that experienced user's could experiment with the fall down menus and decide if to use it.

Dividing Lines

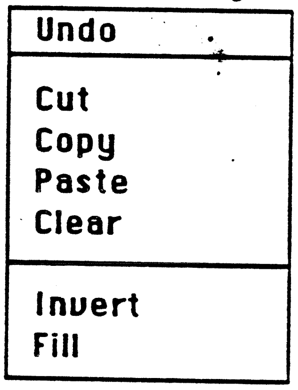
There are two standard ways to partition groups of items from one another. The first is a *dividing line*, selected by an ItemName which is a single dash. It uses the space of an entire item and a whole item record. The second way is a *underline*, defined by setting bit 13 in ItemFlag in the item's record. This will draw a solid line on the bottom most line of the item. The underline doesn't use any more space, on the screen or in memory, than the item would without it..

The disadvantage with an underline is there isn't as much space separating items, which is the dividing line's function.

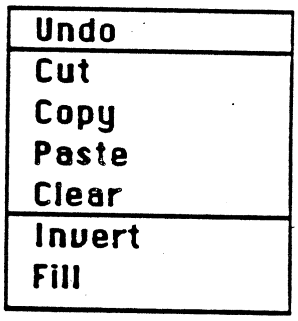
The advantage of an underline is you can get more items in the menu and still have dividing lines. Also, the user would have a shorter distance to go from the menu's title to the last item in the menu, it would save a little memory, and the menu would draw faster.

In the example below are two menus, both showing the same information. Menu A uses dividing lines and has 9 items. Menu B uses underlines and has 7 items. Menu B looks alittle crowded and would look even worse if one of the uderlined items had descending lower case letters.

Menu A - Dividing Lines



Menu B - Underlines



COMPARISON OF CORTLAND MENU CALLS AND MAC'S

These comparisons are in general terms. The biggest changes on the Cortland are; no resources, color, multiple menu bars, resolution, menu bar included into control list. Because of these changes the inputs to Cortland Menu routines are different from the Mac inputs. Further disparity is caused because the difference bewteen the 65816 and 68000, and the way to optimize for each of them. Once it's clear that emulating the 68000 with a 65816 is not the way to go, me may as well take advantage of the 65816 strengths. Also, if something made more sense doing it differently on the Cortland it was done that way. The important thing was that it act like a Mac on the screen.

For porting code from the Mac to the Cortland I think glue routines would be in order. Hopefully, myself, or someone else, will create a library of these glue routines.