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. New calls **CheckItem**, **SetItemMark**, **GetItemMark**, **EnableItem**, **DisableItem**, **NewMenu**, **DisposeMenu**, **SetMenuID**, **SetItemID**, **SetSysBar**, **GetSysBar**, and **InitPalette**.
06/18/86	Fall Down menus removed. **InitMenus**, **BootMmgr**, **MmgrReset**, **MmgrVersion** names changed. Additional parameter, user ID, passed to **MenuStartup** (formerly **InitMenus**). Direct access to menu record no longer supported. Custom menus being rethought, and not currently complete. **GetMenuPtr** and **GetItemPtr** removed. **InsertMenu** and **InsertItem** are being redesigned, and are not complete. Now using standard error return code, although it is always 'No error'.
07/15/86	Changed the term Menu String to menu/item line list. **NewMenu** now allocates only one menu at a time. Special characters in menu/item lines changes; X is now color replace highlighting, ID numbers must be included. **InsertMenu**, **InsertItem**, **DeleteMenu**, **DeleteItem** are complete. Custom menus are defined.
07/16/86	Replacements for pages 8 and 9, I wasn't using proper ID numbers in my examples.
08/13/86	Removed standard color menus. One character has been added to the front of menu and item lines. Added **GetMHandle** and **GetMenuMgrPort** calls. Changed inputs to **SetMenuFlag**. Menu/Item strings may terminate with a zero in addition to a return. Change to menu records.

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 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 the selected title 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 a 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 Bars

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 normal, hilite, or outline color. The color can be used for menus, items, nonsystem menu bars, and the rest of the screen.

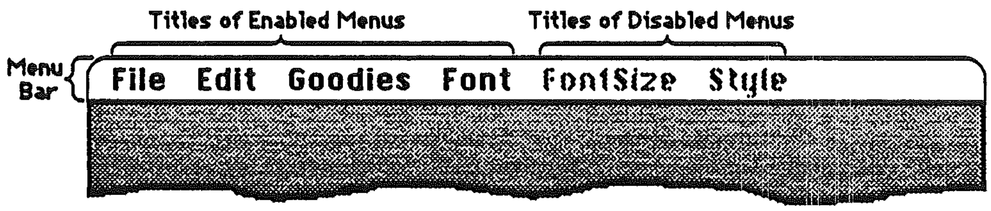
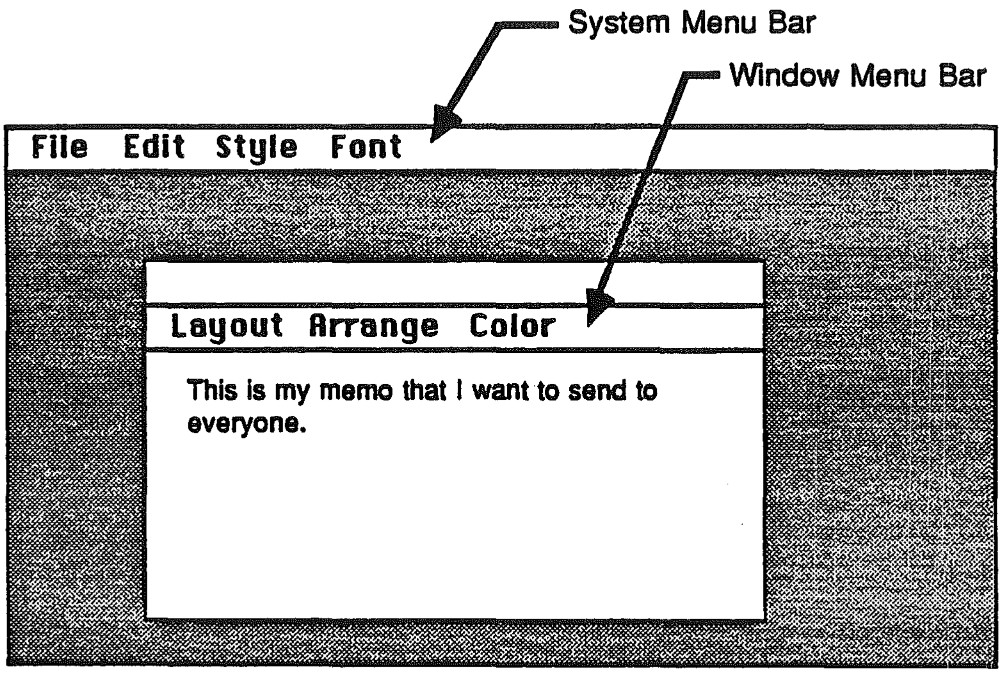


Figure 1. The System Menu Bar

Window Menu Bars

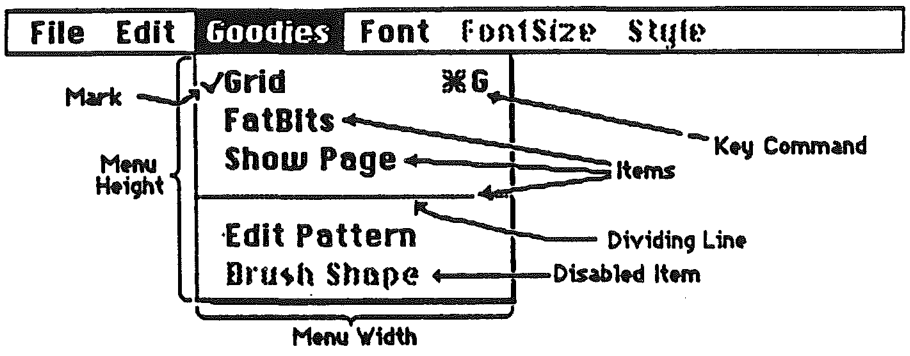
In addition to the System Menu Bar your application can have various window menu bars. These can appear anywhere in windows. Window menu bars are provided to give you more flexibility and to address the limited resolution in 320 mode. Window menu bars should be used moderately, if at all. Window menu bars perform in the same manner as the System Menu Bar.



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 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 (any charcter) 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 character. Pressing indicated character 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 own 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**.

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

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). You can assign one or two keyboard equivalents per item. One equivalent is the primary, and is displayed to the right of the item. The other equivalent is the alternate and is not displayed. The alternate equivalent should be the lower case equal to the primary equivalent (which should be upper case). See **MenuKey** for a discussion on how items are searched.

> Note: For consistency between applications, you should specify an upper case letter as the promary keyboard equivalent.

Using the Menu Manager

To use the Menu Manager, you must have previously initialized QuickDraw. For user interaction you must use the Event Manager. If you are going to be using the Window Manager, it must be initialized before the Menu Manager.

The first Menu Manager routine to call is the initialization procedure, **MenuStartup**. Among other things, **MenuStartup** will create an empty system menu bar and draw it on the screen.

Your program then must define menus and items by providing a list of menu and item lines to **NewMenu** for each menu (see MENU LINES AND ITEM LINES) and **InsertMenu** to add them to the system menu bar. **FixMenuBar** may be of use in setting default sizes.

After created, **DrawMenuBar** will draw the titles of the added menus.

The following section only applies if you are using **TaskMaster** for standard user interaction. If you are not using **TaskMaster**, you will have to perform the following actions:

> To handle user input your application (if not using **TaskMaster**) calls **MenuSelect** or **MenuKey** with a pointer to an extended event record (see **MenuSelect**). From the extended event record, **MenuSelect** and **MenuKey** will extract information, like event position and key states to determine user interaction with the current menu bar.
>
> **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. If using multiple menus you must switch the current menu, by calling **SetMenuBar**, before calling **MenuSelect**. **MenuSelect** will return immediately if the starting position (event position) is not inside the menu bar. If the starting position is in the current menu bar, **MenuSelect** will retain control, and pull down appropriate menus —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 a pointer to an extended event record which

contains the character that was typed and key states. Applications should respond the same way to auto-key events as to key-down events. **MenuKey** will check the event record, and only respond when the Command key was held down in addition to a key being entered.

MenuSelect and **MenuKey** (or **TaskMaster**) will return a result in the TaskData field of the extended event record. The values returned will be:

- Zero in the low-order WORD if there was no selection. In this case there is no further action required and your application should continue to poll the use.

- If the low-order WORD is nonzero, it is the item ID of a selected item. The high-order is athe menu ID of the menu the selected item is in. When a selection is made, the Menu Manager will leave the title of menu highlighted. 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 affected menu title 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 menus and items.

There are several miscellaneous Menu Manager routines that may be of use to applications. **CalcMenuSize** calculates the dimensions of a menu and is called by **FixMenuBar**. **CountMItems** counts the number of items in a menu. **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 Lines and Item Lines

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

```
>>Title 1\N1
-Item string 1\N256
-Item string 2\N257
-Item string 3\N258
.
```

This is a simple list of one menu line and 3 item lines. The first character on a line denotes the start of a menu or an item in a menu. Each line is terminated by a return (decimal 13) or a null byte (0). The character to denote a title is whatever the very first character is in the first line. The character to denote items is the first character on subsequent lines that is different from the title character. And lastly, a character different from the item character, and after the title, denotes the end of the list. In the example, the '>' character is the title charcter, '-' is the item character, and '.' is the terminating character. However, any characters may be used, as long as the title and item characters are different, and the termination character is different from the item character. So, the title and termination character may be the same.

The second character on each line (other than the termination line) is a place holder for the length of the string. NewMenu will replace the second character with the string's length. Therefore, after a NewMenu call the string data will have been altered.

> Note: The menu/item string must stay in their original memory. NewMenu sets pointers in the menu record to the address of the strings.

In the example you'll notice a backslash, '\', followed by a 'N' and a number. The backslash denotes the end of a title's text and the beginning of special characters. The 'N' is a special character that precedes an ID number. A decimal, unsigned, ASCII ID number immediately follows. Every menu title and item must have an ID number, even dividing lines. The ID number for each menu title should be different from every other menu on a menu bar. The ID of an item should be different from every other item on a menu bar. Items that are dividing lines, and always disabled, can have the same ID number.

Special characters are:

\	Beginning of special characters.
*	Followed by a primary, then an alternate character to be used as a keyboard equivalents. Use a space for no alternate character.
B	Bold the text.
C	Followed by a character to be used to mark the item.
D	To dim (disable).
H	Hexidecmal, nonASCII, ID number follows, low byte/high byte.
I	Italize the text.
N	Decimal ASCII ID number follows, any length, between 1 and 65535.
U	Underscore the text.
V	Places a dividing line under the item without using a separate item.
X	Use color replace, and not XOR, highlighting.

All the special characters pertain to items. Special characters *, B, C, I, U, and V do not pertain to menu titles.

An example of a menu and item lines using mulitple special characters and different title, item, and terminating characters:

$Title 1\N1	Title character is '$', ID = 1, can be same as an item's.
Item string 1\N256*Xx	Item character is a space, ID = 256, key eqivalents X and x.
Item string 2\BC√UN257	Item character is a space, bold, checked, underscored, ID = 257.
Item string 3\IN258	Item character is a space, text will appear italized, ID = 258.
$	Terminating character, can be the same as the title character.

Some more special stuff. Using just the @ symbol in a title will give you the Apple logo. An example of an Apple logo menu title:

$@\N1X	Apple logo title, ID = 1, color replace highlighting.

Note: The X special character (color replace highlighting) should always be used with the Apple logo.

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

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

A single dash, '-', for an item's text will denote a dividing line. Special characters apply to dividing lines. Dividing lines should always be marked as dimmed, 'D'.

ID Number Assignment

ID numbers are not assigned automatically, it must be assigned in the menu/item line list. Item ID numbers are allocated accordingly:

$0000	0	Internal use, generally means front, or first item in menu.
$0001 - $00FF	1-255	Reserved for desk accessory items.
$0100 - $FFFE	256-65534	Reserved for application use.
$FFFF	65535	Internal use, generally means end, or last item in menu.

Menu ID numbers are allocated accrodingly:

$0000	0	Internal use, generally means front, or first menu in bar.
$0001-$FFFE	1-65534	Reserved for application use.
$FFFF	65535	Internal use, generally means end, or last menu in bar.

What ID numbers to use? Here are two suggestions for schemes in ID number assignment. The first is to number menus from 1 to n, and items from 256 to 256+n. The item ID can then be used to index into a table of selection handling routines when a selection is made. The other scheme is to use the lower WORD of the handling routine's address as the ID. Different items still must have different ID numbers, so NOPs at the head of the routine could be used for different entry points and ID numbers.

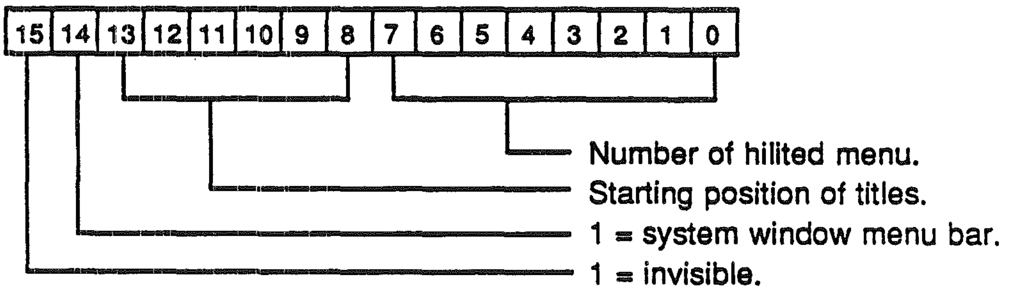
ID numbers can later be changed to anything you would like with calls to, SetItemID, GetItemID, SetMenuID, and GetMenuID.

Menu Records

The Menu Manager keeps all the information it requires for its operations on a particular menu bar in a **menu bar record**. The record contains the menu's position, color, menu lists, item lists, and other flags the Menu Manager needs to manage menus. The menu bar record is the same as a control record.

NextCtrl	LONG	Handle of next control.
CtrlOwner	LONG	Window menu belongs to, zero for system menu bars.
CtrlRect	RECT	Coordinates of menu bar.
CtrlFlag	WORD	Defined below.
CtrlValue	WORD	Not used, should be zero.
CtrlProc	LONG	$0A000000.
CtrlData	LONG	TRUE if system window's menu, FALSE if application's.
CtrlRefCon	LONG	Reserved for application's use.
CtrlColor	LONG	Pointer to color table, defined below.
MenuList	LONG[]	Array of menu handles, zero terminates list.

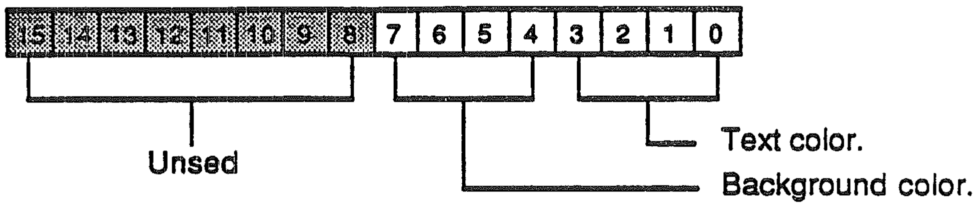
Menu Bar - CtrlFlag:



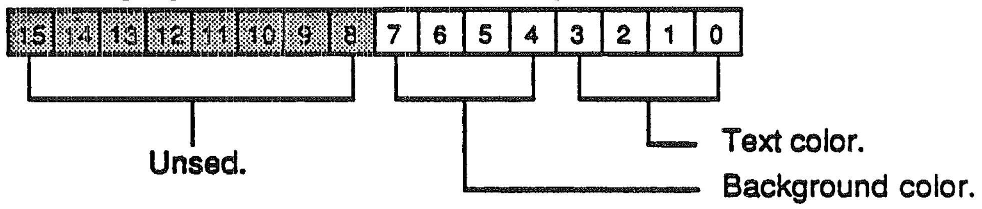
* *Invisible flag is not yet implemented.*

Menu Bar Color Table:

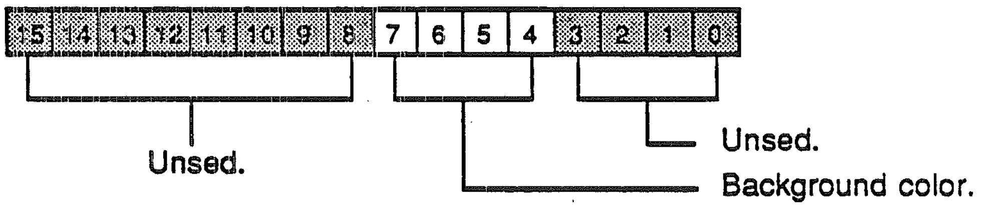
Color 0 - unhighlighted color of text and background:



Color 1 - highlighted color of text and background:



Color 2 - color of outline:

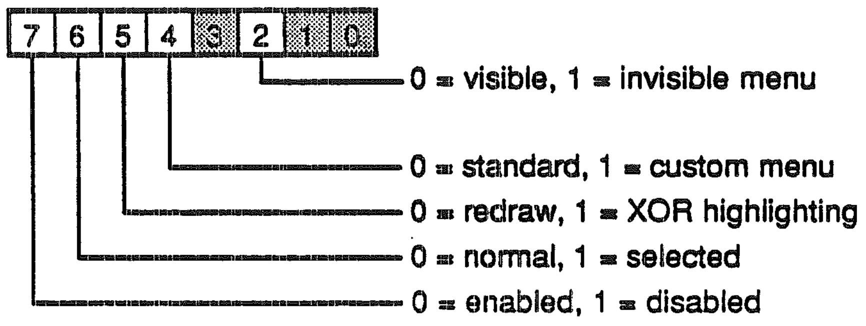


The MenuList is an array of menu handles in the menu bar. Menus records are only partially defined. Only the first half of the record is definded.

MenuID	WORD	Menu's ID number.
MenuWidth	WORD	Width of menu.
MenuHeight	WORD	Height of menu.
MenuProc	LONG	Pointer to menu definition procedure, zero for standard menu.
MenuFlag	BYTE	Defined below.
MenuRes	BYTE	Reserved.
FirstItem	BYTE	Reserved.
NumOfItems	BYTE	Reserved.
TitleWidth	WORD	Width of title.
TitleName	LONG	Pointer to title text, first byte equals length.

(the rest of record is not defined)

MenuFlag


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

* *Invisible flag is not yet implemented.*

Not defining menu records completely has good and bad sides. Access to menu information will be slower if calls to the Menu Manager have to be made. However, the delay would have to be measured in miliseconds, and the delay never seen on the screen. On the plus side, future Menu Managers would not be tied to an older, possibly inadequate, record structure. The chances of improving the current Menu Manager, and maintaining compatibility accross future hardware, is greatly improved by allowing records to change.

Item records not defined at all for standard menus.

Defining Your Own Menus *(not completed)*

The standard type of menu is predefined for you. However, you may want to define your own type of menu—one with more graphics, or perhaps a nonlinear text arrangement. QuickDraw and the Menu Manager make it possible for you to do this.

To define your own type of menu, you write a menu definition procedure. The Menu Manager calls the menu definition procedure to perform basic operations such as drawing the menu.

To create a custom menu record you will have to allocate a block of memory large enough for your menu record. Only the defined part (see MENU RECORDS) of the menu record has to follow Menu Manager form, the rest of the record is up to you. Another way is to pass a menu line with no items to NewMenu and then resize the allocated block to your needs. Fields in the menu record that need to be initialized are:

MenuID	Menu's ID number.
MenuWidth	Width of menu, or you can wait for the mSize.
MenuHeight	Height of menu, or you can wait for the mSize.
MenuProc	Pointer to menu definition procedure.
MenuFlag	In addition to other flags, bit 4 must be set.
TitleWidth	Width of title.
TitleName	Pointer to title text, first byte equals length. Or some other data you wish.

The Menu Definition Procedure *(not completed)*

You may choose any name you wish for the menu definition procedure. The inputs and outputs are:

inputs:	message:WORD	Operation to perfrom.
	theMenu:LONG	Handle of menu.
	RectPtr:LONG	Pointer to RECT enclosing menu.
	xHitPt:WORD	X coordinate of point to check.
	yHitPt:WORD	Y coordinate of point to check.
	param:WORD	Addition parameter for each operation.
output:	Result:WORD	Depends on operation.

The message parameter identifies the operation to be performed. It has one of the following values:

mDrawMenu	= 0	Draw the menu.
mChoose	= 1	Tell which item was chosen and highlight it.
mSize	= 2	Calculate the menu's dimensions.
mDrawTitle	= 3	Draw the menu's title.
mDrawItem	= 4	Highlight or unhighlight an item.
mGetItemID	= 5	Return item's ID number.

The parameter theMenu indicates the menu that the operation will affect. RectPtr is the rectangle (in global coordinates) in which the operation is to be performed.

mDrawMenu

The message mDrawMenu tells the menu definition procedure to draw the menu inside the RECT pointed to by RectPtr. The current grafPort will be the Menu Manager port. The standard menu definition procedure figures out how to draw the menu items by looking in the menu record at the data that defines them. For menus of your own definition, you may set up the data defining the menu items any way you like. You should also check the enableFlags field of the menu record to see whether the menu is disabled (or whether any of the menu items are disabled, if you're using all the flags), and if so, draw it in gray. You may even print the items in a different font, as long as you restore the original when you finish. Returned value is not used.

mChoose

When the menu definition procedure receives the message mChoose, the yHitPt/xHitPt parameter is the mouse location (in global coordinates), and the param parameter is the item number of the last item that was chosen from this menu (param is initially set to 0). The procedure should determine whether the mouse location is in an enabled menu item, by checking whether yHitPt/xHitPt is inside the RECT pointed to by RectPtr, whether the menu is enabled, and whether yHitPt/xHitPt is in an enabled menu item:

- If the mouse location is in an enabled menu item, unhighlight param and highlight the new item (unless the new item is the same as the param), and return the item number of the new item.

- If the mouse location isn't in an enabled item, unhighlight param and return zero.

Note: When the Menu Manager needs to make a chosen menu item blink, it repeatedly calls the menu definition procedure with the message mChoose, causing the item to be alternately highlighted and unhighlighted.

mSize

Finally, the message mSize tells the menu definition procedure to calculate the horizontal and vertical dimensions of the menu and store them in the menuWidth and menuHeight fields of the menu record. Returned value is not used.

mDrawTitle

When the menu definition procedure receives the message mDrawTitle when the title of the menu must be drawn. The param parameter is FALSE to draw the title as unhighlighted, and TRUE to draw as highlighted. The RECT pointed to by RectPtr encloses the title area. Return FALSE to have the Menu Manager draw the title, if the title is a text string pointed to by TitleName. Return TRUE if you complete the operation.

mDrawItem

The mDrawItem command is a request to draw an item in its highlighted or unhighlighted state. If param is positive it is the item number and should be draw draw as unhighlighted. If param is negative it is the negated form of the item number and should be drawn as highlighted. This command is given when the user makes a selection and is used to blink the selection.

mGetItemID

Param equals the item's number and the definition procedure is asked to return the item's ID number. The item number is the value returned by mChoose.

Dividing lines vs. Underlines

Dividing Lines

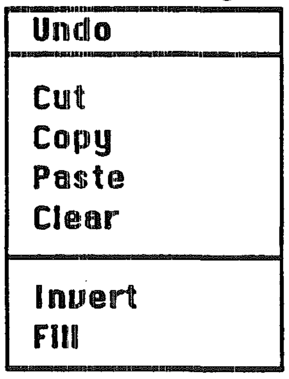
There are two standard ways to partition groups of items from one another. The first is a *dividing line*, selected by an item title which is a single dash. It uses the space of an entire item and a whole item record. The second way is a *underline*, set either in the menu line, or SetItemFlag. This 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.

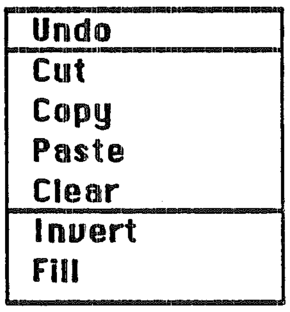
The advantage of an underline is you can get more items in the menu and still have dividing lines. Also, the user would have a shorter distance to go from the menu's title to the last item in the menu, it would save a little memory, and the menu would draw faster.

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


Undo
Cut
Copy
Paste
Clear
Invert
Fill