Custom Menus

How to make a custom menu handler

This topic discusses how to create non-standard menus. For instance, you can use a **custom menu** for multiple-row palettes or tool icons, or to display text in a font other than Chicago.

The menuProc field of the MenuInfo is normally set to 0, indicating the standard menu definition routine. To install a custom routine, you can do one of:

- Create an 'MDEF' resource having the code of the routine. Store its resource ID into offset 8 of a 'MENU' resource. When you use **GetMenu** to read the 'MENU' resource, the 'MDEF' is read and installed automatically.
- Read an 'MDEF' resource using **GetResource** and store its handle into MenuInfo.menuProc.
- Use the following technique to create a Handle leading to a routine within your application. Store that handle into MenuInfo.menuProc.

Here's a formal description of the routine you need to write:

#include <Menus.h> **Menu Manager**

pascal void **MyMenuDefFn(***msg*, *theMenu*, *menuRect*, *hitPt* , *itemID* **);**

short	*msg* ;	0=draw; 1=choose; 2=calc size; 3=calc popup
MenuHandle	*theMenu* ;	handle leading to menu of interest
Rect	***menuRect* ;	rectangle enclosing menu (global coordindates)
Point	*hitPt* ;	mouse position, used in "choose" message
short	***itemID* ;	address of item ID incoming and outgoing

This user-defined pascal-type function performs the low-level handling of custom menus. It is responsible for calculating the size of the menu rectangle, displaying the menu, and taking care of highlighting items and determining which item was selected.

msg is a function code. It specifies which of four operations to perform. and will always be one of the following values, defined in Menus.h:

mDrawMsg (0)	Draw the entire menu within *menuRect*.
mChooseMsg (1)	Unhighlight menu item ***itemID*, highlight the item associated with mouse point *hitPt* , and store the item ID of that item in ***itemID*. This message is passed repeatedly while the menu remains pulled down.
mSizeMsg (2)	Calculate the size of the menu and store values in the menuHeight and menuWidth fields of *theMenu*.
mPopUpMsg (3)	Calculate the rectangle in which the popup menu should appear.
mDrawItemMsg (4)	
mCalcItemMsg (5)	

theMenu is a handle leading to a variable-length MenuInfo structure. Use (*theMenu)->*fieldName* to access the fields of the structure.

menuRect is the address of an 8-byte Rect structure, expressed in the global coordinates of the Window Manager's grafPort.

hitPt is a mouse position, expressed in global coordinates. This parameter is used when *msg* =mChooseMsg. It is also used (oddly) in the mPopUpMsg message.

itemID is the address of a short that identifies an item in the menu. It contains the most recent value you stored there in a previous call (or 0 before the first call). Items are numbered starting at 1. When the application's call to **MenuSelect** returns, this value is incorporated in the return value.

Returns: none

Notes: This is a pascal-type function and must be declared as such. See the example, below.

While your Menu Definition Function is active, the Window Manager's grafPort is current. If you change any fields of the GrafPort (e.g., by changing the active font), be sure to change them back before returning.

MDEF Handle Tricks

While developing an application, it gets tiresome to work with code resources. Here's a handy way to keep a menu definition function inside your application. The trick involves creating a handle to a 6-byte data area and storing a JMP instruction there that you initialize to point to your code. Since this constitutes self-modifying code, you should flush the '040 cache or the jmp instruction will sit in the copyback cache and the resource won't contain the correct code.

```
#define _CacheFlushTrap 0xA0BD

pascal void myMenuDefFn( MenuHandle myMenu, Handle  mDefHandle,
    short *intPtr);    /* predefine the function */

myMenu=GetMenu( MENU_ID );     /* read menu resource */
InsertMenu( myMenu );     /* install in menu list */

mDefHandle=NewHandle( 6 );  /* allocate 6 bytes */
intPtr=(short *)*mDefHandle;  /* address those bytes */

*intPtr++=0x4ef9;  /* a JMP instruction */
*intPtr++=HiWord(myMenuDefFn);  /* where to jump */
*intPtr=LoWord(myMenuDefFn);
(*myMenu)->menuProc = mDefHandle;  /* attach to MenuInfo */

if (TrapAvailable( _CacheFlushTrap))   /*Cache must be flushed on */
                                       /* 68040 machines since*/
                                       /* code is self-modifying */
        asm
        {
            dc.w _CacheFlushTrap
        }
    CalcMenuSize( myMenu ); /* must do this explicitly */
```

Note that this is completely safe, since the handle's data (the JMP instruction) can float around when memory gets compacted. However, be sure that the menu definition function code is locked in memory (just have it in the main code segment). This also works for internally-defined window definition and list definition functions. (See **Compatibility Guidelines** for a definition of the **TrapAvailable** routine.)

Also note that **CalcMenuSize** must be called explicitly since it is normally called implicitly by **GetMenu** and other functions such as **SetItem**, that change a menu's size.

Messages

The Menu Manager remains responsible for nearly all services (most of which are list-management operations). Your function is responsible only for drawing the menu; highlighting and unhighlighting items and passing back the current item number; and calculating the size of the menu rectangle.

Drawing the Menu: Your menu can have anything from icons to pictures, to patterns, to text in any font or style, arranged in any format. When called with *msg* =mDrawMsg, you should draw the interior of the menu. The pull-down frame has already been drawn and filled with white. The pixels behind the menu have been saved on the heap.

To learn about the contents of the menu, you can use any of the **GetItem**Xxx functions or you can access the extended MenuInfo structure, or you might just define the menu contents within the code itself.

Calculating the Menu's Size: When *msg* =mSizeMsg, you should determine the size of the menu and store its width and height in the menuWidth and menuHeight fields of the MenuInfo structure addressed by *theMenu* . This message is passed early, since the Menu Manager needs to know how large an area needs to be saved before it pulls the menu down.

Use **CountMItems** if you don't know how many items are in the menu. Use **StringWidth** to learn the width, in pixels, of a string of text.

Popup Menu Position: The mPopUpMsg message is sent when the application calls **PopUpMenuSelect**. Your handler should calculate global coordinates for the rectangle in which the menu should be drawn. Store the result into *menuRect*. Your calculations should yield a rectangle such that: when the menu it drawn, *itemID* 's top-left corner will align with *hitPt*.

> **Note**: In this case, the fields of *hitPt* will contain the left and top coordinates for an item rectangle. These coordinates are **in reverse order of a Point** structure. Use *hitPt*.h as the top and *hitPt*.v as the left coordinates.

Choosing (highlighting) Items: This is probably the most complicated of the four operations. This message is passed repeatedly to your MenuDefFn as the mouse moves while the menu is visible. You are responsible for showing which item is pointed to by the mouse and you must pass back its item number.

When *msg* =mChooseMsg, you must determine which item is associated with *hitPt* (call this "mouseItem"). If mouseItem is already highlighted, just set **itemID* to that item number.

If mouseItem is disabled or invalid (e.g., *hitPt* is not inside of *menuRect*), then unhighlight item number **itemID* and set **itemID* to 0.

If mouseItem is enabled and valid, then unhighlight **itemID* (the previous item), highlight mouseItem and store its value into **itemID* . (Note: If **itemID* =mouseItem on entry, you can assume that the correct item is already highlighted).

When the user releases the mouse button, and the last item returned in **itemID* is non-zero, the Menu Manager causes that item to blink (by sending mChooseMsg several times, alternating between *hitPt*=(0,0), and *hitPt*=the mouse position when the button was released). The application's call to **MenuSelect** (that caused all of this to happen) returns with the low word of the return value set to the same value as the last value you stored in **itemID*.

There is a bug in the Menu Manager (as described in Apple Technote 222) that prohibits the menu item from flashing correctly if the menu record contains no text. The workaround is to put dummy text in the menu record. Be sure to add this dummy text to all menu items in your custom 'MDEF' that have no text in the menu record.

Low-Level Hooks

For even more control over a **custom menu**, there are two global variables that hold addresses of low-level "Hook" routines that you may intercept.

MenuHook	(0xA2C) Called repeatedly after a menu has been pulled down while the mouse button is pressed. No parameters.
MBarHook	(0xA30) Called after the menu title is highlighted but before the menu is drawn. It receives one parameter on the stack-the address of the rectangle enclosing the menu to be drawn. Return with register D0 set to 0 to continue normally; set D0 to 1 to abort the call to **MenuSelect**.

Both variables are normally set to 0, indicating use of the default routines. To intercept one, store the address of your custom routine into the variable.

The following is an example of a **custom menu** definition routine. It is incomplete since it calls several "black-box" functions that are not included and it assumes menu items are exactly 12 dots high and 50 dots wide. However, it gives the basic outline of how the function should look and act.

Example

```
#include <Menus.h>

pascal void myMenuDefFn(short msg, MenuHandle whichMenu, Rect *menuRect,
        Point hitPt, short *itemID )
{
    short itemCount, j, mouseItem;

    itemCount = CountMItems( whichMenu );

    switch (msg) {
    case mDrawMsg:
        for ( j=1;  j<=itemCount;  j++ )
            myDraw1Item( whichMenu, menuRect, j ); /* draw one */
        break;

    case mChooseMsg:
        mouseItem = myFindItem( hitPt, menuRect );
        if (mouseItem==0) {                    /* out of bounds or disabled */
            myUnhilite( menuRect, *itemID );
            *itemID=0;                          /* return "cancel" code */
        }
        else if ( mouseItem != *itemID ) {
            myUnhilite( menuRect, *itemID );    /* unHilight previous */
            myHilite( menuRect, *mouseItem );   /* hilight new */
            *itemID = mouseItem;                /* return new */
        }
        break;

    case mSizeMsg:
        (*whichMenu)->menuWidth=50;
        (*whichMenu)->menuHeight=itemCount*12; /* assume 12 pts high */
        break;

    case mPopUpMsg:
        menuRect->top = hitPt.h - ((*itemID-1) * 12);
        menuRect->left = hitPt.v;
        menuRect->right = menuRect->left+50;
        menuRect->bottom = menuRect->top+(itemCount*12);
        break;

    }   /* end of switch */
}   /* end of myMenuDefFn */
```