New Technical Notes

Macintosh

Developer Support

Menu Manager Q&As

Toolbox M.TB.MenuMgr.Q&As

Revised by: Developer Support Center October 1992
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As and Q&As revised this month are marked with a bar in the side margin.

OK to call GetMenu repeatedly except for 128K or 512K Mac
Written: 8/13/90
Last reviewed: 8/1/92

Why does Inside Macintosh Volume I, page 352, warn about calling GetMenu only once for a particular menu?

—

Inside Macintosh warns about using GetMenu more than once because in the original (64K) ROMs GetMenu would load the 'MENU' resource when first called and then it would use the 'MDEF' ID to load the proper 'MDEF', and then store the handle to the 'MDEF' inside the 'MENU' record. The problem was that the second time you called GetMenu it would not check to see if the menu had been previously loaded, and would assume that the high byte of the 'MENU' record held the ID of the 'MDEF' to load; when it actually held the high byte of the handle to the 'MDEF'.

This problem was fixed in the 128K Macintosh Plus ROMs and all subsequent systems. Unless your application runs on a 128K or 512K Macintosh, you are safe to call GetMenu

repeatedly.

Forcing a pop-up menu to be drawn with extra space on right side
Written: 11/16/90
Last reviewed: 8/1/92

How can I force a Macintosh pop-up menu to be drawn with extra space on the right side, as figure 3 of Human Interface Note #9 shows?

—

You need to get the latest version of the PopupMenu Sample code. You can obtain this from the Developer's CD, or from the Developer Services files on AppleLink. You will also need the Utilities file in the same folder as the sample code. These should get you going in the right direction.

Detecting if a menu belongs to a Macintosh Desk Accessory

Written: 12/21/90
Last reviewed: 1/16/91

How do I detect that a menu that is up belongs to a Desk Accessory (DA)?

—

The following code is one method (note that the menu will be the last one in the list):

```
FUNCTION daMenu: BOOLEAN;

CONST
    MenuListLoc = $A1C;

TYPE
    MenuRec = RECORD
        menuoH: MenuHandle;
        menuLeft: INTEGER;            { Left edge of Menu. }
    END;    { MenuRec }

    MenuList = RECORD
        lastMenu: INTEGER;            { Offset to last MenuHandle. }
        lastRight: INTEGER;           { Right edge of last Menu's title. }
        notUsed: INTEGER;
        mArray: ARRAY[0..0] OF MenuRec;
    END;    { MenuList }

    MenuListPtr = ^MenuList;
    MenuListHdl = ^MenuListPtr;

VAR
    MLHdl: MenuListHdl;
    nbrMenusX6, numMenus: INTEGER;
    daMenuHdl: MenuHandle;

BEGIN

    MLHdl := MenuListHdl(MenuListLoc);
    nbrMenusX6 := ORD(MLHdl^^.lastMenu);
    numMenus := nbrMenusX6 DIV 6;
    daMenuHdl := MLHdl^^.mArray[numMenus - 1].menuoH;
    daMenu := (daMenuHdl^^.menuID < 0);

END;    { daMenu }
```

Help, Keyboard, and Application menu IDs don't change

Written: 3/25/91

Last reviewed: 8/1/92

The System 7.0 Help, Keyboard, and Application menus at the right of the Macintosh menu bar don’t have text titles that can be included in a command string like “File” or “Edit” can be. Do these menus always have the same menu ID? Do you have any scripts we can test with the new Script Manager?

—

The menu ID numbers of the Help, Keyboard, and Application menus are always the same, so the menus can always be identified by their IDs:

```
kHMHelpMenuID = -16490;
kPMKeyBdMenuID = -16491;
kPMProcessMenuID = -16489;
```

While we do not have any scripts available for the Script Manager menu, you can set the Always Show Icon bit of the 'itlc' resource in the system.

Technique for changing a Macintosh menu title

Written: 6/5/91
Last reviewed: 8/1/92

How do I change the title of a menubar menu after it has been installed?

—

Although the format of menu resources is defined in Inside Macintosh I-364, there are no Toolbox calls to change the menu title.

The format used to store menu titles and item names is a packed C string list, where the first byte is the length of the title, followed by the characters in the title and immediately followed by the length of the first item. It would be difficult and _extremely_ unsupported to try to grow this data structure (for example, if the programmer didn’t allocate it), but there is an easier way to solve your problem.

All you have to do is have a separate menu for each title. You can make the menus up on the fly, or just get them all from your resource file at the start of your program. You can then call InsertMenu and DeleteMenu followed by DrawMenuBar to add and remove them from your application’s menubar.

Associating Macintosh color icons with menu items

Written: 7/1/91
Last reviewed: 8/15/91

Is there a way to display Macintosh 'ics8' icons in menus, like the Finder does?

—

You can have color icons associated with menu items, and it’s even quite simple. To

associate a color icon with a menu item, simply make a 'cicn' resource with the proper item number for your icon and the system will use it instead of the 1-bit icon. You cannot associate 'ics8's with menu items yet, however, since this mechanism does not work properly yet.

Code for drawing to a Macintosh window while menu is open

Written: 7/25/91
Last reviewed: 8/1/92

How do I draw to a Macintosh window if it's partially obscured by a menu that's pulled down?

—

The Menu Manager is, by design, graphically the most modal component of the system. The reason for this harkens back to the days of the Macintosh Plus (or the modern time Macintosh Classic) These 68000-based machines do not have the horsepower to modify the visRgns of all visible windows during the menu draw. (It can slow the machine down something fierce if you have too many windows open.) Since the user tends to move back and forth on the menu bar, this could cause a great lag time for menu drawing. The down side, the menu's being almost completely modal, is considered to be a small price to pay.

Fortunately, it is not hard to work around this little gottcha, and I think that with the code snippet below, you'll be able to have the most non-modal menus in existence. Also, the code can lend a pretty cool animated real-time look to your application.

What you have to do is patch the mBarHook as well as the MenuHook. The MBarHook is called right before any menu is drawn, and it is passed the rectangle that the menu is going to go into (minus the drop shadow). All you need to do is to manage the cliprgn of the window that you want to draw into at mBarHook time and then all your drawing during menuHook time will be properly clipped to the menu's interior, allowing the illusion of drawing "behind" the menu. Since this is called only before a menu is drawn, you do have to assume that the cliprgn of the window going into mBarHook is not right; simply call ClipRect(window^.portRect) to ensure that you are starting with a clean clipRgn. Also remember to reset the clipRgn after the call to menu select so that you don't have persistent unsightly "menu holes" in your window when you redraw it.

```
; Here is a sample mBarHook & MenuHook that allows you to safely draw to
; a window while a menu is pulled down
;
; Install the hooks right before you call MenuSelect
; remove them right after just because its the tidy thing to do like this...
   lea    MyMBarHook,A0         ; get our hook routine
   Move.L  A0,MBarHook
   lea    MyMenuHook,A0
   Move.L  A0,MenuHook
   CLR.L  -(SP)                 ; space for MenuSelect
   MOVE.L  Where(A6),-(SP)      ; Mouse coordinates
   _MenuSelect                  ; pass MenuSelect's result
   Move.L  #0,D0
   Move.L  D0,MBarHook
   Move.L  D0,MenuHook
;
; Here is the code for the mBarHook. This routine is called once before any
; menu is pulled down. Its Pascal interface looks something like this:
;     Procedure MBarHook(menuRect:Rect);
; You return a result code in D0, either 0 (continue with menu select)
;                                    or 1 (Abort MenuSelect)
; Since this is called only before a menu is drawn, and not after a menu is
; released, we must always assume that we have been called before when fudging
; the window cliprgn
;
```

```
MyMBarHook  Proc
     Move.L  4(sp),D0  ; get the rectangle ptr off the stack
     MoveM.L  A0-A5/D1-D2,-(sp) ; save some registers
     Move.L  CurrentA5,A5       ; make sure A5 is correct
     Move.L  G.theWindow,A4     ; get the pointer to our window from globals
```

```
     Move.L  D0,A1              ; move the rectPtr into a proper reg
     lea    MyRect,A2           ; now copy the rectangle into a local rect
     Move.L  (A1)+,(A2)+  ;
     Move.L  (A1)+,(A2)+ ;
     clr.L  -(sp)               ; preserve the menuMgrPort
     Move.L  sp,-(sp)           ; on the stack for later
     _GetPort      ;
     Move.L  A4,-(sp)           ; and use our window
     _SetPort
     pea    MyRect        ; convert the passed menu rect to local coord's
     _GlobalToLocal
     pea    MyRect+4
     _GlobalToLocal
     lea    MyRect,A3           ; insetRect(myRect,-1,-1);
     subQ  #1,(A3)
     subQ  #1,2(A3)
     AddQ  #2,4(A3)             ; adjust for drop shadow of menu as well
     AddQ  #2,6(A3)
     pea    portRect(A4)  ; reset the ports cliprect to the whole window
     _ClipRect
     clr.L  -(sp)         ; and turn the adjusted menu rect into a region
     _NewRgn
     Move.L  (sp),A3
     pea    MyRect
     _RectRgn
     Move.L  ClipRgn(A4),-(sp) ; Sutract menu rect from window clip region
     Move.L  A3,-(sp)
     Move.L  ClipRgn(A4),-(sp)
     _DiffRgn                   ; leaving the result in the clip region
     pea    portRect(A4)        ; erase the window for our demo
     _EraseRect
     Move.L  a3,-(sp)           ; get rid of the temporary storage region
     _DisposeRgn
     _SetPort                   ; and put the saved menuMgrPort back
     MoveM.L  (sp)+,A0-A5/D1-D2 ; restore the save registers
     Move.L  (sp)+,D0           ; get the return address
     Move.L  D0,(sp)       ;discard the rect and restore the return address
     moveQ #00,D0               ; signal that we want to continue
     RTS                        ; and back to the menu manager
MyRect    ds.W  4                ; temporary storage
     EndP
;
; And here is the menu hook routine that is called all the time while a menu
; is pulled down you can setport to the window and draw whatever you like.
; (I am just inverting for effect
;
MyMenuHook  PROC
     MoveM.L  A0-A5/D0-D2,-(sp)
     Move.L  CurrentA5,A5
     Move.L  G.theWindow,A4
     clr.L  -(sp)
     Move.L  sp,-(sp)
     _GetPort                   ; get the current grafport
     Move.L  A4,-(sp)
     _SetPort
     pea    portRect(A4)
     _InverRect                 ; draw in the window behind the menu
     _SetPort
     MoveM.L  (sp)+,A0-A5/D0-D2 ; restore the save registers
     RTS
     EndP
```

Dimming Macintosh menu items from a custom MDEF

Written: 8/2/91
Last reviewed: 8/1/92

What is the correct method for a custom MDEF to dim menu items when running under System 7? What is the preferred method for determining whether to draw in a gray color or paint the items with a gray pattern?

—

The proper method for dimming text in menu items is to use the GrayishTextOr transfer mode when drawing text. This is documented on page 17-17 of Inside Macintosh Volume VI. This is what Apple uses. this mode takes into account both color and black and white screens. A simple method for dimming non-text items is to set the OpColor to gray and then to draw the non-text item in Blend mode.

TheMenu global and determining when a menu is pulled down
Written: 8/15/91
Last reviewed: 8/1/92

According to Inside Macintosh, the global variable TheMenu at address 0xA26 contains the menu ID of the currently highlighted menu, but TheMenu is always zero when MenuSelect is highlighting a menu. My test routine inserts its address in the global variable MenuHook to give me the value of the currently highlighted menu by looking at the global variable TheMenu. What am I missing?

—

Documentation on the global variable TheMenu is unclear. The variable is supposed to contain the menu whose title is currently highlighted after a call to MenuSelect. The standard 'MBDF' does not update this variable while it is selecting from a menu, only after a selection is made. The bottom line is that TheMenu is not valid at MenuHook time.

If there is some reason that you need to know when a specific menu is pulled down you might be able to use a custom MDEF (see the latest Developer CD Series disc for sample code) and then you would know which menu was pulled down, or you could use the mBarHook, which is called only when a menu is about to be drawn. You might then be able to use the rectangle that is passed to it to help you discover which menu is pulled down, by running through the current menu bar until you found the menu with the matching rectangle.

Using Shift and Option keys with menu Command keys
Written: 9/16/91
Last reviewed: 8/1/92

How can I use Shift and Option keys with my menu Command keys?

—

There is no standard way of having shifted or optioned Command keys in menus.

First, using the Option key is a problem because holding down the Option key changes the meaning of the key pressed (for example, Option+8 = •, and Option+u is a dead key that prepares for an umlaut to be added to the following character). These dead keys and "optional" characters differ between international versions and even move around the keyboard on various international keyboards. Under System 7.0 it is now possible to install several international

scripts on the same Macintosh, so the Option key mappings could change frequently within the same document.

Second, although Apple recommends standard mnemonic Command-key equivalents for the major menu items, it is very difficult for command+œ to be mnemonic for most users, especially when it is just one of many such combinations. This would, of course, be considered a severe interface violation by users :). Likewise, the distinction between Command-F and Command-f is lost on most users, as an “F” is still an “f.”

The only way to do this kind of extended Command-key mapping is to write your own MDEF code and write a custom version of MenuKey. DTS strongly recommends against this. It is quite likely that if your users need this many menu keys that some elements of your interface could be better moved off the menu bar and into some other interface method.

Moving a menu bar from one screen to another
Written: 10/16/91
Last reviewed: 8/1/92

Under certain conditions, I want to have the menu bar moved from the main Macintosh screen to the second screen if there is one. Is this possible? Would you please give me some thoughts?

—

We don’t have a way to move the menu bar from one screen to another under application control.

It’s rather contrary to our human interface guidelines; the user expects the menu bar to stay in one place. The user can control the placement of the menu bar using the Monitors control panel.

Even if the user changes the screen on which the menu bar appears, the change isn’t effective until restart. There are several data structures that are affected by the menu bar placement, and they can’t really be changed “on the fly” because the system doesn’t even know where all of them are. These include the visible regions of other applications’ windows that may underlay the menu bar. System extensions like AppleShare (which displays an activity indicator in the menu bar, to the left of the apple menu title) may also assume that the menu bar is immobile.

Also, users assume that most new windows and alerts will appear on the main screen (ie, the one with the menu bar). While some applications may want color windows to show up on the “deepest” monitor, even this should be controllable as a preference item.

For your application, it might be more flexible, understandable, and useful to have a tool palette within your document’s window or in a floating window that the user can move at

will.

No way to include a color icon with a Macintosh menu title
Written: 10/33/91
Last reviewed: 8/1/92

How do we include color icons in menu titles, similar to balloon help and MultiFinder? Do we create an MDEF, use balloon help's technique, or is there another method?

___

There's no good solution for including color icons in menu titles. To begin with, an MDEF draws the contents of the menu but not the title itself. Furthermore, System 7 eliminated all support for accomplishing what you want. (How to write an MBDF, which is what is needed, hasn't been documented.) Currently it isn't possible to have an icon with the title of a menu, unless you are the system.

Long menus: Enabling or disabling >31 items
Written: 12/11/91
Last reviewed: 12/11/91

Is there an easy way to to enable and disable more than 31 menu items in my Macintosh popup menu, short of writing a new MDEF? If not, please point me to code that demonstrates how to write such an MDEF.

—

The behavior you require isn't supported by the standard System MDEF because only 32 bits are allocated for the entire menu to control disabling items. This limitation was probably considered reasonable, since applications rarely have long menus with distinct items on them. Generally, the only menus that long are things like Font menus with a similar options, which are enabled or disabled as a group, without need for individual control.

You'll have to write your own MDEF to do this, with a custom mechanism to manage enabling and disabling items. A good MDEF sample, demonstrating many features, is the Concordia sample, which can be found on the latest Developer CD Series disc in Tools & Apps (Moof!):OS/Toolbox:Menu Defproc 1.0.3. Another useful sample is the source to the System MDEF, which is also available on the Developer CD in Tools & Apps (Moof!):OS/Toolbox:DefProcs:mdefproc.a.

You might also consider changing your interface. A menu that long with so many distinct options can create a cumbersome interface for your users.

Enabling & disabling >31 Macintosh menu items
Written: 1/22/92
Last reviewed: 2/28/92

My application can have a menu that contains more than 31 menu items. I have read that menus that have more than 31 menu items can't have those items whose value is greater than 31 be controlled by DisableItem and EnableItem. What's interesting is that when I call DisableItem to disable the entire menu, those menu items whose value is greater than 31 are disabled. Is there some way to control the status of menu items whose value exceed 31?

—

The behavior you describe is standard for the Menu Mangler. Calling DisableItem on 'item'

0 is a special case that causes *all* items in a given menu to be affected, regardless of its size. This is the behavior of the standard text MDEF; the enable/disable state of individual items is managed by the other 31 bits in the flags word.

The only way to enable or disable things in menus with more than 31 items (“even if it were advisable to create such things,” she said, feeling the warm breath of the Human Interface Police watching over her shoulder), is to write your own custom MDEF and manage the state information yourself. For information on writing your own MDEF, take a look at pages 362-365 of Inside Macintosh Volume I, pages 248-255 of Inside Macintosh Volume V, the

Macintosh Technical Note “Parameters for MDEF Message #3,” and the sample Custom MDEF on the current Developer CD Series disc.

Changing fonts & font sizes displayed in Macintosh pop-up menus

Written: 10/19/90
Last reviewed: 8/1/92

How can I change fonts and font sizes displayed in Macintosh pop-up menus?

—

One way to get pop-up menus in any font and style is to use the Communications Toolbox. There’s a pop-up menu 'CDEF' included with it that has a resource ID of 63. You can tell it to use the font and size of the current port’s font. You can get documentation on this in the Macintosh Communications Toolbox v.1.0 from APDA (part #M0232LL/D). The Comm Toolbox is an integral part of System 7, so you know that this 'CDEF' is always around if you’re running 7.0 or later. Your users would need to install the Comm Toolbox to use your product under System 6.

You can also trick the standard 'MDEF'. Just before calling PopUpMenuSelect, patch the TextFont trap so that it points to a routine of yours that replaces the specified font number (which will be on the stack) with the font number for the font you want, then it calls the normal TextFont routine. When MenuSelect returns, patch TextFont back to what it was before you patched it. The patch routine looks something like the following if, for example, you want the menu to use the Geneva font:

```
MyTextFont PROC

          MOVE.W   #$0003,4(SP)   ;Replace system font spec with Geneva spec
          MOVE.L   NormTrap,-(SP) ;Put addr of normal TextFont onto the stack
          RTS                     ;Call normal TextFont

          ENDPROC
```

The routine to install this patch will look something like this:

```
VAR
 NormTrap : LongInt; {Address of normal TextFont routine}
 WMPort   : GrafPtr; {Pointer to Window Manager port}
 CurrPort : GrafPtr; {Pointer to current GrafPort}
 CurrFont : Integer; {Font number of normal font for Window Manager Port}
 CurrSize : Integer; {Size of normal font for Window Manager port}

{Save current GrafPort for restoring later}
GetPort (CurrPort);

{Set the current port to the Window Manager port and set its font size}
GetWMgrPort (WMPort);

{Save the normal font and size and set the new size}
CurrFont := WMPort^.txFont;
CurrSize := WMPort^.txSize;
SetPort (WMPort);
TextSize (9);

{Patch TextFont so that our MyTextFont routine will be called.}
{NormTrap is global so that MyTextFont can see and use it.}
```

```
NormTrap := NGetTrapAddress ($A887, ToolTrap); {$A887 is trap for TextFont}
NSetTrapAddress (MyTextFont, $A887, ToolTrap);
```

```
{Now call PopUpMenuSelect}
SetPort (CurrPort);
MenuChoice := PopUpMenuSelect (MyMenuHandle, MyMenuLoc.v, MyMenuLoc.h, 1);

{Reset the TextFont trap to what it was before we changed it}
NSetTrapAddress (NormTrap, $A887, ToolTrap);

{Set the current font back to Chicago 12}
SetPort (WMPort);
TextSize (CurrSize);
TextFont (CurrFont);

{Set the current port back to what it was}
SetPort (CurrPort);
```

This routine assumes that the standard 'MDEF' will ALWAYS call TextFont, although this is a rather dubious assumption. The routine works on both Color QuickDraw and non-Color QuickDraw machines because the standard 'MDEF' copies the WMgrPort text characteristics to the CWMgrPort.

There's yet another way to trick the standard 'MDEF'. You can change the QuickDraw bottleneck for drawing text so that it also changes the font. It works similarly to patching TextFont, but because the standard 'MDEF' does NOT copy the bottlenecks to the CWMgrPort on a Color QuickDraw machine, you must check for Color QuickDraw yourself and then use the appropriate port and type of QDProcs record (either QDProcs or CQDProcs). In the following example, assume there's no Color QuickDraw:

```
VAR
 WMPort    : GrafPtr;    {Pointer to Window Manager port}
 MyProcs   : QDProcs;    {My bottlenecks}
 CurrProcs : QDProcsPtr; {Pointer to normal bottlenecks}
 CurrPort  : GrafPtr;    {Pointer to current GrafPort}
 CurrFont  : Integer;    {Font number of normal font for Window Manager Port}
 CurrSize  : Integer;    {Size of normal font for Window Manager port}

{Save current GrafPort for restoring later}
GetPort (CurrPort);

{Set the current port to the Window Manager port and set its font size}
GetWMgrPort (WMPort);

{Save the normal font, size, and procs and set the new size}
CurrFont := WMPort^.txFont;
CurrSize := WMPort^.txSize;
CurrProcs := WMPort^.grafProcs;
SetPort (WMPort);
TextSize (9);

{Set our bottleneck text proc to point to our text proc}
SetStdProcs (&MyProcs);
MyProcs.textProc := @MyTextProc;
WMPort^.grafProcs := @MyProcs;

{Now call PopUpMenuSelect}
SetPort (CurrPort);
MenuChoice := PopUpMenuSelect (MyMenuHandle, MyMenuLoc.v, MyMenuLoc.h, 1);

{Set the bottleneck back to normal}
WMPort^.grafProcs := CurrProcs;

{Set the current font back to Chicago 12}
SetPort (WMPort);
TextSize (CurrSize);
```

```
TextFont (CurrFont);

{Set the current port back to what it was}
SetPort (CurrPort);

PROCEDURE MyTextProc (byteCount : INTEGER; textBuf : Ptr;
                                          numer, denom: Point);

 VAR
   currPort : GrafPtr;
   currFont : Integer;
   currSize : Integer;

BEGIN
 GetPort (currPort);
 currFont := currPort^.txFont;
 currSize := currPort^.txSize;
 TextFont (1);
 StdText (byteCount, textBuf, numer, denom);
 TextFont (currFont);
 TextSize (currSize);
END;
```

Both these methods of “tricking” the standard 'MDEF' need global variables. The Macintosh Tech Note “Stand-Alone Code, *ad nauseam*” covers global use with stand-alone code.

The last way to do what you’re trying to do (and what the Comm Toolbox does) is set the SysFontFam ($0BA6) global to the font number of the font you want, set the SysFontSize ($0BA8) global to the font size you want, and then set the long word at LastSpExtra ($0B4C) to -1. The Font Manager caches the system font and only occasionally goes to the SysFontSize and SysFontFam globals when it needs to draw the system font. When LastSpExtra is set to -1, it acts as a flag to the Font Manager to flush the font cache and load the system font defined by the SysFontFam and SysFontSize globals.

So which method would I use? If my program had to run under all systems, I’d use this last method. If my program had to run under System 7 only, I’d use the Comm Toolbox’s 'CDEF'.

DA code for inserting and removing a menu

Written: 2/28/92
Last reviewed: 4/7/92

I am working on a DA that adds a menu to the System MBAR. Under System 7.0.1 with 32-bit addressing turned off, the menu works reliably. Turn on 32-bit addressing, and releasing on a selected menu item drops you into a debugger. I never get to respond to the AccMenu event inside the DRVRControl. Do you have a source sample that shows how to insert and remove a menu that works across the 24/32-bit barrier?

—

There is no difference when inserting a menu in a DA when 32-bit addressing is turned on. Here is a modified version of the menu sample that ships with MPW which has a menu and all the code needed to install or remove the menu at the proper time. It works equally as well under System 7 and 6 as well as with 32-bit addressing on or off. Some other problem probably is causing you to crash.

```
---------------------------- Memory.c code ----------------------]
/*
```

```
File Memory.c
Copyright Apple Computer, Inc. 1985-1987
All rights reserved.

* Memory - report the amount of free space in the
* application and system heap, and on the boot volume.
*
* This is the sample C desk accessory. The desk accessory does not
* use any global variables. Instead, it allocates a handle to a
* structure that holds some "global" variables. This sample program
* could be written without having to use this structure, but then it
* wouldn't be as informative...
*/
#include    <types.h>
#include    <osutils.h>
#include    <memory.h>
#include    <devices.h>
#include    <menus.h>
#include    <events.h>
#include    <quickdraw.h>
#include    <fonts.h>
#include    <windows.h>
#include    <files.h>
#include    <errors.h>
#include    <toolutils.h>
#include    <packages.h>

/*
* Macro to compute owned resource id
*/

#define OWNEDRSRCID(id)    (0xC000 | (((-(id)) - 1) << 5))

/*
* String constant indexes for STR# resource
*/

#define    APPHEAP        1
#define    SYSHEAP        2
#define    DISK           3
#define    FREEON         4

#define    ACCEVENT       64
#define    ACCRUN         65
#define    ACCMENU        67

/* This structure type holds the global variables used by this
* desk accessory */

typedef struct {
   int     rsrcID;        /* Computed rsrc id of STR# and WIND resources */
   Str255    strBuf;      /* Buffer to read strings into */
   MenuHandle MyMenu;     /* our menu in the bar... */
   short    Installed;    /* our menu flag... */
} Globals;

pascal short DRVROpen(CntrlParam *ctlPB, DCtlPtr dCtl)
{
   #pragma    unused (ctlPB)
   GrafPtr        savePort;
   WindowPeek    myWindow;
   long        heapGrow;

   /*
    * If the windowPtr is non-nil, we already have a window open.
```

```
    * This desk accessory ignores multiple opens.
    */
   if (dCtl->dCtlWindow != nil)
       return noErr;

   GetPort(&savePort);

   /*
    * Get a handle to some storage that will hold our pseudo-global
    * variables. Save the handle in a location accessible by
    * all the driver routines.
    */
   dCtl->dCtlStorage = NewHandle(sizeof(Globals));
   /*
    * Compute the resource id of the owned 'STR#' resource that
    * contains all of the program's text strings.  The id is saved
    * in one place that can be accessed by all the driver routines.
    */
   ((Globals *)(*dCtl->dCtlStorage))->rsrcID =
                                          OWNEDRSRCID(dCtl->dCtlRefNum);

   /*
    * wStorage = nil (allocate on the heap)
    * visible = false, behind = -1, goAway = true, refCon = 0
    */
   myWindow = (WindowPeek)GetNewWindow(((Globals *)
                     (*dCtl->dCtlStorage))->rsrcID, nil, (WindowPtr) -1);
   /*
    * Set windowKind to the DA refNum, which is negative.
    */
   myWindow->windowKind = dCtl->dCtlRefNum;
   /*
    * Store the windowPtr in the Device Control Entry
    */
   dCtl->dCtlWindow = (WindowPtr)myWindow;
   /*
    * Now compact the heap in the most violent way.
    * Purge whatever's purgeable.
    */
   (void) MaxMem(&heapGrow);
   ((Globals *)(*dCtl->dCtlStorage))->MyMenu =
                       GetMenu(((Globals *)(*dCtl->dCtlStorage))->rsrcID);
   InsertMenu(((Globals *)(*dCtl->dCtlStorage))->MyMenu,0);
   ((Globals *)(*dCtl->dCtlStorage))->Installed = 1;
   DrawMenuBar();
   SetPort(savePort);
   return noErr;
}

void    AddTheMenu(Globals *globals)
{
   if (globals->Installed == 0)
     {
       InsertMenu(globals->MyMenu,0);
       globals->Installed = 1;
       DrawMenuBar();
     }
}

void    RemoveTheMenu(Globals *globals)
{
   if (globals->Installed == 1)
     {
       DeleteMenu(globals->rsrcID);
       globals->Installed = 0;
```

```
        DrawMenuBar();
      }
}

pascal short DRVRPrime(CntrlParam *ctlPB, DCtlPtr dCtl)
{
   #pragma    unused (ctlPB, dCtl)
   return noErr;            /* Not used in this desk accessory */
}

pascal short DRVRStatus(CntrlParam *ctlPB, DCtlPtr dCtl)
{
   #pragma    unused (ctlPB, dCtl)
   return noErr;            /* Not used in this desk accessory */
}

pascal short DRVRControl(CntrlParam *ctlPB, DCtlPtr dCtl)
{
   extern    void    doCtlEvent();
   extern    void    doPeriodic();

   /*
    * The current grafPort is saved & restored by the Desk Manager
    */
   switch (ctlPB->csCode) {
       case ACCEVENT:                  /* accEvent */
           HLock(dCtl->dCtlStorage);  /* Lock handle since it will be
                                         dereferenced */
           doCtlEvent( *((EventRecord **) &ctlPB->csParam[0]),
                      (Globals *)(*dCtl->dCtlStorage));
           HUnlock(dCtl->dCtlStorage);
           break;

       case ACCRUN:                /* periodicEvent */
           doPeriodic(dCtl);
           break;

       case ACCMENU:
           SysBeep(1);
           break;

       default:
           break;
   }
   return 0;
}

static void doCtlEvent(register EventRecord *theEvent, Globals *globals)
{
   register    WindowPtr    myWindow;
   extern        void        drawWindow();

   if (theEvent->what == updateEvt) {
       myWindow = (WindowPtr) theEvent->message;
       SetPort(myWindow);
       BeginUpdate(myWindow);
       drawWindow(myWindow, globals);
       EndUpdate(myWindow);
   }
   if (theEvent->what == activateEvt) {
       if ((theEvent->modifiers & activeFlag) != 0 )
           AddTheMenu(globals);
       else
           RemoveTheMenu(globals);
   }
```

```
}

static void doPeriodic(DCtlPtr dCtl)
{
   extern void drawWindow();

   SetPort(dCtl->dCtlWindow);
   HLock(dCtl->dCtlStorage);    /* Lock handle since it will
                                   be dereferenced */
   drawWindow(dCtl->dCtlWindow, (Globals *)(*dCtl->dCtlStorage));
   HUnlock(dCtl->dCtlStorage);
}

/*
* Display the contents of the window.
* The current port is assumed to be set to the window.
*/
static void drawWindow(WindowPtr window, Globals *globals)
{
   THz                 saveZone;
   Str27                volName;
   HVolumeParam        myParamBlk;
   void                printNum(unsigned long);
   static    StringPtr    text(int index, Globals *globals);

   if (window == nil)
       return;        /* "can't happen" */

   TextMode(srcCopy);
   TextFont(monaco);
   TextSize(9);

   MoveTo(6, 10);
   TextFace(bold);

   saveZone = GetZone();

   DrawString(text(APPHEAP, globals));
   SetZone(ApplicZone());
   printNum(FreeMem());

   DrawString(text(SYSHEAP, globals));
   SetZone(SystemZone());
   printNum(FreeMem());

   SetZone(saveZone);

   DrawString(text(DISK, globals));
   myParamBlk.ioNamePtr = volName;
   myParamBlk.ioVRefNum = 0;        /* Boot volume */
   myParamBlk.ioVolIndex = 0;
   (void) PBHGetVInfo((HParmBlkPtr)&myParamBlk, false);
   printNum((unsigned long)myParamBlk.ioVAlBlkSiz * myParamBlk.ioVFrBlk);

   DrawString(text(FREEON, globals));
   TextFace(underline);
   DrawString(volName);
}

static void printNum(unsigned long num)
{
   unsigned    char    numStr[32];

   TextFace(normal);
   NumToString(num, numStr);        /* Its possible that a large unsigned
```

```
                                        will come back negative! */
   DrawString(numStr);
   TextFace(bold);
}

pascal short DRVRClose(char *ctlPB, DCtlPtr dCtl)
{                          /* Save & Restore current grafPort? */
   #pragma    unused (ctlPB)
   WindowPtr window;

   if (((Globals *)(*dCtl->dCtlStorage))->Installed == 1)
     DeleteMenu(((Globals *)(*dCtl->dCtlStorage))->rsrcID);
   DisposeMenu(((Globals *)(*dCtl->dCtlStorage))->MyMenu);
   window = (WindowPtr) dCtl->dCtlWindow;
   if ( window != nil) {
       dCtl->dCtlWindow = nil;
       DisposHandle(dCtl->dCtlStorage);
       DisposeWindow(window);
   }

   return 0;
}

static StringPtr text(int index, Globals *globals)
{
   GetIndString(globals->strBuf, globals->rsrcID, index);
   return(globals->strBuf);
}

------------------------------ memory.r ------------------------------
/*
* File Memory.r
*
* Copyright Apple Computer, Inc. 1985-1987
* All rights reserved.
*
* Sample desk accessory resource file.
* This incorporates the DRVR header information (defined here)
* with the linked code (stored as a 'DRVW' resource in the link command)
*/

#include "Types.r"                /* To get system types */
#include "MPWTypes.r"            /* To get 'DRVW' type */

#define DriverID    12

#ifdef NOASM_BUILD
/*
* This will produce a DRVR resource from the special DRVW type.
* (this eliminates the need for using the Assembler to create
*    the DA header, but makes it impossible to use SADE, oops!)
*
* Note that the ID 12 is irrelevant, since the Font/DA Mover
* will renumber it to something else when installing it anyway.
*
* The leading NUL in the resource name is required to
* conform to the desk accessory naming convention.
*
* The resource is declared purgeable.    If the code were to
* do funky things like SetTrapAddress calls (requiring the code to
* be around at all times), we would have to set it nonpurgeable.
*/

type 'DRVR' as 'DRVW';            /* Map 'DRVW' => 'DRVR' */
```

```
resource 'DRVR' (DriverID, "\0x00Memory", purgeable) {
   /*
    * DRVR flags
    */
   dontNeedLock,      /* OK to float around, not saving ProcPtrs */
   needTime,          /* Yes, give us periodic Control calls */
   dontNeedGoodbye,   /* No special requirements */
   noStatusEnable,
   ctlEnable,         /* Desk accessories only do Control calls */
   noWriteEnable,
   noReadEnable,
   5*60,              /* drvrDelay - Wake up every 5 seconds */
   updateMask,        /* drvrEMask - This DA only handles update events */
   0,                 /* drvrMenu - This DA has no menu */
   "Memory",          /* drvrName - This isn't used by the DA */
   /*
    * This directive inserts the contents of the DRVW resource
    * produced by linking DRVRRuntime.o with our DA code
    */
   $$resource("Memory.DRVW", 'DRVW', 0)
};
#endif

/*
* Since desk accessories cannot use global data (and the C compiler
* considers string constants to be global data) and we really don't
* want to hard-code strings in our source, the strings used by the
* DA are stored in the resource file. Note the expression used to
* figure out the resource id.
*/

resource 'STR#' (0xC000 | (DriverID << 5), "Memory's Strings") {
   {
       "AppHeap: ";
       "  SysHeap: ";
       "  Disk: ";
       " free on "
   };
};


resource 'WIND' (0xC000 | (DriverID << 5), "Memory's Window") {
   {322, 10, 338, 500},
   noGrowDocProc,
   visible,
   goAway,
   0x0,
   "Free Memory (# Bytes)"
};

resource 'MENU' (0xC000 | (DriverID << 5), preload) {
   (0xC000 | (DriverID << 5)), textMenuProc,
   0xffffffff,    /* Disable dashed line, enable About and DAs */
   enabled, "MemDA",
   {
       "Beep one time",
           noicon, nokey, nomark, plain;
       "Beep Twice",
           noicon, nokey, nomark, plain
   }
};

----------------------------------- build script ----------------------
Set Echo 1
C Memory.c -o Memory.c.o -r {SymOptions}
```

```
Asm DAEntry.a -o DAEntry.a.o
Link {SymOptions} -w -da -rt DRVR=12 ∂
   -m DAEntry -sg Memory             # DAEntry is located in DAEntry.a.o ∂
   DAEntry.a.o                       # This must preceed DRVRRuntime.o ∂
   "{Libraries}"DRVRRuntime.o        # This must preceed Runtime.o ∂
   Memory.c.o ∂
   "{Libraries}"Runtime.o ∂
   "{Libraries}"Interface.o ∂
   -o Memory -c DMOV -t DFIL

-------------------------------- DAEntry.a --------------------------
********************************************************************************
****
****    DESK ACCESSORY and DEVICE DRIVER Entry Code/Data
****
********************************************************************************

       STRING    PASCAL

       INCLUDE 'ToolEqu.a'
       INCLUDE 'SysEqu.a'

       CASE OBJ
************************** DESK ACCESSORY ENTRY ************************
       IMPORT %DRVRMain

DAEntry Proc Export                                  ; See Device Manager IM:2
;
; First we need to set the drvrFlags (IM II-188), choose from
;
;    dReadEnable   enable driver for read operations      (drivers only)
;    dWritEnable   enable driver for writing              (drivers only)
;    dCtlEnable    enable driver/da for control operations
;    dStatEnable   enable driver/da for status operations (drivers only)
;    dNeedGoodBye  driver/da needs a "goodbye kiss"
;    dNeedTime     driver/da needs "main thread" time
;    dNeedLock     driver will be accessed at interrupt level (drivers only)
;
   DC.B      (1<<dCtlEnable) + (1<<dNeedTime) ; periodic, control flags set
   DC.B      0                                        ; Lower byte is unused
;
; Next is the drvrDelay (IM II-188), set only if dNeedTime flag set above
;
   DC.W        5*60                                   ; 5 sec periodic update
;
; Next is the the drvrEMask (IM I-444), which events DA can respond to...
; Must be NIL for drivers, for DA's choose from
;    mButDwnEvt        mouse button down is event 1
;    keyDwnEvt         key down is event 3
;    keyUpEvt          key up is event 4
;    autoKeyEvt        auto-repeated key is event 5
;    updatEvt          update event
;    activateEvt       activate/deactive event
;
   DC.W        (1<<updatEvt)               ; Handle activate, update events
;
; Next is the the drvrMenu (IM I-444), Menu ID of DA's menu, or NIL
;
   DC.W        -16000                                  ; No associated menu
;
; Next are the offsets to the main routines of the driver/DA
;
   DC.W        %DRVRMain - DAEntry                ; Open routine
   DC.W        %DRVRMain - DAEntry +4             ; Prime - unused for DAs
   DC.W        %DRVRMain - DAEntry +8             ; Control
```

```
   DC.W        %DRVRMain - DAEntry +12            ; Status - unused for DAs
   DC.W        %DRVRMain - DAEntry +16            ; Close

;
; Next are the offsets to the main routines of the driver/DA
;
DAName
   DC.B        'Memory'                        ; DA/DRVR Name
   ORG         DAName+32                       ; Pad string out to 32 bytes

   END
```

Macintosh tear-off menus
Written: 5/8/91
Last reviewed: 8/1/92

Where can I find documentation describing the official way to implement tear-off menus in my Macintosh application?

—

If you want to implement tear-off menus, you can use MacApp 3.0 or later, or follow the article in MacTutor entitled “Tear-off Menus & Floating Palettes” in the ‘Best of’ MacTutor volume 4, page 112. The best approach is to use MacApp 3.0.

The second method, described in MacTutor, is not supported by Apple, though developers used this method to implement tear-off menus on their own before MacApp 3.0

Lining up Macintosh submenus and 'SICN' resources
Written: 10/29/90
Last reviewed: 6/10/91

What would cause one of the hierarchical menus in a set of Macintosh menus including both small icons ('SICN's) and hierarchical menus to pop up in the wrong place?

—

The problem you have noticed in submenus not lining up when you mix them with 'SICN's is true and is fixed under System 7.0. To work around this for the current release of System 6, you need to minimize the mixing of 'SICN's with hierarchicals. If you need to mix 'SICN's and submenus you may have to write your own MDEF, which does not have this problem. Although I do not recommend writing your own MDEF, sample source to a standard MDEF is available on developer CDs and on AppleLink.

How to add icons to System 7.0 menus
Written: 3/6/91
Last reviewed: 6/19/91

How do I add icons to System 7.0 menus?

—

You can still add icons to menus in System 7.0 using ResEdit 2.1. To do this,simply open up the menu resource that you want to play with, open up the particular menu ID that you want to edit, and notice that there is a “Choose Icon...” option in the MENU menu.

Use dialog scrolling list instead of long menu list
Written: 5/17/90
Last reviewed: 6/27/91

Can the standard Macintosh MDEF handle menus with more than 255 items? If not, do you have a simple fix to the standard MDEF source to increase this maximum?

—

The standard MDEF has a hard limit of 255 items (and a limit that only the first 31 can be enabled and disabled). To handle more than this number you will have to write your own MDEF.

Apple advises against writing an MDEF that handles more than 255 items because the speed of menu drawing, scrolling, and highlighting will tend toward unacceptably slow. Also, selecting from a >50-item menu is a very unpleasant experience for the user.

A better method of selecting among large numbers of items such as fonts is to allow the user to select from a scrolling list of fonts presented in a dialog. Key presses in the dialog scroll the list to match (á la the file list in Standard File) Finally, allow the users to customize their font menus, preferably both for the application's default and additionally by document. Examples of most of the above methods are seen in MacDraw II and Microsoft Word.

This does not rule out other methods that you may have thought of to relieve the user of waiting through the scroll of hundreds of items. If you would like to flesh out your ideas, Link: MacInterface and we will be happy to help you develop your ideas.

Code for appending a menu item containing meta-characters
Written: 7/24/92
Last reviewed: 9/15/92

We use AppendMenu to create pop-up menus of user-defined objects. The objects are named by the user, and we store the names as Cstrings. We have a "menu resource stub" which we load, append to, and subsequently dispose of, when the user clicks in the appropriate location. However, there's a problem with this strategy. AppendMenu strips meta-characters, which it recognizes as instructions for determining menu characteristics such as command key equivalents or styles. Is there any way in which we can ask AppendMenu to ignore such characters ?

—

The following code lets you append a menu item containing meta-characters:

```
menu = GetMHandle(mSomeMenu);      // mSomeMenu = menu ID of some menu in your application.
AppendMenu(menu, "\psome text");   // Append a menu item with no meta-character.
```

```
SetItem(menu, CountMItems(menu), "\psome / text");
    // Replace the menu item we added with no meta-character with
    // text that has a meta-character.
```

SetItem doesn't analyze for meta-characters, so first use AppendMenu with some dummy text, and then use SetItem to replace that text. That ought to do it.