Chapter 15 Articles & Notes

Macintosh One-Liners

by Eric Pepke

The Macintosh One-Liners are intended to condense into a small space information about some of the most common Macintosh problems and programming pitfalls. Each one-liner is a single line of text, shorter than 80 characters, which informs about one aspect of Macintosh use or programming.

One-liners give either facts or advice. The facts may be obvious to some people and obscure to others but are important for all. The advice is intended to help keep people from running into the most common nontrivial problems. Like proverbs, the advice may not be absolute and may sometimes be more conservative than is strictly necessary. However, I have found that a little constructive paranoia can go a long way toward avoiding problems, and more than once I have taken a precaution which seemed extreme at the time but which saved my skin later on.

The one-liners started as a list I made for myself of things to remember while writing programs. I have augmented them with my condensed records of several years of Info-Mac, Usenet, and Delphi digests and one year of Usenet reading. People who have contributed to the list since its first release are mentioned at the end. The result is very much a gestalt of the Macintosh lore I have seen and depends on the wisdom and efforts of many people. If I have forgotten to include your name, I apologize.

The one-liners are organized into two sections. The first lists the one-liners under broad categories without explanation. This is good for cutting out and sticking on the wall. The second section gives some extra information on each one-liner.

All references given are to Apple documentation. In some cases, the references explain the problem, but in most, they just describe the parts of the Toolbox you need to use. Most of the problem fixes are a result of my experience and the experience of those people who are listed below. The acronym IM means Inside Macintosh; the page numbers refer to the Addison-Wesley trade paperback version. TN means Technical Note. I use the HyperCard stack, available from sumex-aim.stanford.edu, apple.com, and no doubt other places as well. I have avoided referring to any third-party software tools and only refer to a minimal set of Apple tools. This is not to endorse Apple in any way. It's just that I don't want to get into the product comparison business. I use a minimal set of ubiquitous tools, and I describe fixes in terms of those tools. If you have different tools, you can translate. I have also tried to avoid using language-specific statements where possible, but all my example code is in C. Again, you can translate.

Compiled by Eric Pepke
Additional material by Steve Maker, Keith Rollin, Gregory Dudek, Brian Bechtel,
Henry Minsky, Carl C. Hewitt, Jim Lyons, Alex Lau, Kent Borg, Peter W. Poorman,
Ross Yahnke, Mark Fleming, Mark Anderson.

Send suggestions to pepke@gw.scri.fsu.edu on the
Internet or PEPKE@FSU on BITNET. Have fun.

The Main Loop and Events
Call MaxApplZone and MoreMasters when the application starts up.
If you call SetApplLimit, do it before calling MaxApplZone.
Use HT in MacsBug while running to estimate how many times to call MoreMasters.
Don't use SetEventMask to disable mouseUp events. Better not to use it at all.
SetPort to a known good grafPort once every time through the event loop.
Calling WaitNextEvent with more than 50 ticks will fail on some systems.
Set the cursor on suspend and resume events.
Call GetDblTime to get the maximum time for a double click.
Measure double click time from mouse up to mouse down.
Call either WaitNextEvent or both GetNextEvent and SystemTask.
Call IsDialogEvents and DialogSelect even if GetNextEvent returns false.

Menus
Use SetItem to include meta characters literally in menus.
Never make MENU resources purgeable.

Resources
GetResource never produces resNotFound. Check for a NIL handle instead.
To create a resource file, call Create, then CreateResFile.
To open a resource file read-only for shared access, use OpenRFPerm.
Don't leave ResLoad set to false.
GetResource on a dctb may return a non-resource copy of the dctb.

Windows, Alerts, and Dialogs
Drag windows to the bounding box of GetGrayRgn.
Hide scroll bars when deactivating a window.
Call DrawGrowIcon when activating or deactivating a window with a grow region.
DrawGrowIcon does not check to see if the window has a grow region.
Call PenNormal before calling DrawGrowIcon.
itemHit will not be set when a dialog filter is called.
Use a disabled UserItem to draw the roundrect outline around the OK button.
ModalDialog assumes the dialog is already visible and in the front.
Use screenBits . bounds to center dialogs, alerts, etc. below the menu bar.
If you save window locations in files, they may not be valid for all monitors.
DragWindow expects startPt in boundsRect; if not it may not call SelectWindow.
SelectWindow does not automatically call SetPort. You must do that yourself.
DialogSelect responds to activate events but ignores suspend/resume events.
Call PenNormal before calling DrawControls.
The Control Manager only works when the origin of a window is at (0, 0)
To find the position of a window, use LocalToGlobal on the origin.

Drawing
Always set the VisRgn and ClipRgn of offscreen ports.
Set the ClipRgn first when making a picture.

Don't make rowBytes in bitMaps greater than 8191.
To dim text, draw a rectangle with penPat=gray and penMode=patBic over it.
To draw rotated text, draw to an offscreen bitmap, rotate it, and CopyBits it.
Don't use picSize to determine the size of a picture. Check the handle size.
Never draw outside a window except in XOR mode for temporary effects like drag.

To avoid animation flicker, synchronize drawing to the vertical retrace.
Lock handles to pictures before calling DrawPicture.
CopyBits on more than 3Kb data will work, but it might have to allocate memory.
The small Mac screen is 512 pixels wide by 342 pixels high including menu bar.

Interrupts and VBL Tasks
Don't call any Memory Manager routines during an interrupt.
Unlocked handles may not be valid during an interrupt.
To synchronize to the vertical retrace on Macs with slots, use SlotVInstall.

Files
Don't write in the application file. This will fail with read-only devices.
Use PBGetVInfo to convert a VRefNum to a volume name.
Delete uses the Poor Man's Search Path, so don't delete blindly.
File Manager routines with dirID=0 use the Poor Man's Search Path.
Truncate and reallocate files before overwriting to reduce fragmentation.
Check/change the creator and type of Save As... files before overwriting.
If you rewrite files by deleting and creating, copy all Finder information.
Directory ID's are longs, not shorts. Shorts work ALMOST all the time.
If a file version number appears in a file manager call, always set it to 0.
To convert a pathRefNum to a name or file number, use PBGetFCBInfo.
Prevent the creation of files with names that begin with a period.
Write/update the Finder info before writing data or resource forks.

Handles and Pointers
Lock handles before passing their dereferenced pointers to any routine.
Lock handles before setting referenced data to expressions containing functions
Put an odd long at location zero on a 68000 to help find NIL handle references.
Call MoveHHi before locking a handle to avoid memory fragmentation.
Use HGetState/HLock/HSetState to lock a handle then put it back as it was.

General
Always use unsigned characters within text and Pascal-format strings.
Save application preferences in a folder named Preferences in the System Folder
Use SysEnvirons to find the System (Blessed) Folder.
Use GetAppParms to get the name of the application.
The high bit of SysParam . volClik enables the alarm clock.
Check the application name at $910 before exiting with ES within MacsBug.
To exit to shell in the mini-debugger, enter SM 0 A9 F4 and then G 0.
In Pascal, don't nest procedures to be passed by procedure pointer.
In Pascal, "with theHandle^^" is unsafe if memory compaction can occur.

Controversy Corner (Don't shoot me; I'm just the messenger.)
Avoid writing tail patches for traps.
There is no official way to tell if MultiFinder is running or not.

[*This section was created by Eric after my futile attempt at trying to explain his one-liners. I showed him my idea and he was real interested and expanded on it. The idea was that his one-liners was very helpful at remembering the correct things to do but to help some beginning programmers , I thought that an explanation would help them understand why. Also I added a few comments and they are shown by square brackets. MXM*]

Call MaxApplZone and MoreMasters when the application starts up.
References: IM II-30, IM II-31, TN 53

A call to MaxApplZone is needed for an application to get as large a heap as it can under all systems. Usually, this is what you want to do. Call MoreMasters early in the application to get enough blocks of master pointers for the anticipated number of relocatable memory blocks to be allocated. [You should also call MoreMasters from code segment 1 and not from your intialization segment.]

If you call SetApplLimit, do it before calling MaxApplZone.
References: IM II-30

MaxApplZone expands the application zone to the heap limit set by SetApplLimit. It doesn't make any sense to call MaxApplZone first. Most applications will not need SetApplLimit at all, but the few that do should call it before MaxApplZone.

Use HT in MacsBug while running to estimate how many times to call MoreMasters.
References: IM II-22, IM II-31, MacsBug 6.0, TN 53

There is an art to esimating how many times to call MoreMasters. If you call it too few times, the Memory Manager will be forced to call it later on in the application, which can fragment the heap. If you call it too many times, you will be wasting master pointers that you do not need. It's usually better to call MoreMasters too many times than too few. To estimate the number of times, you need to have some idea of how many relocatable blocks your application typically allocates. A good way of finding out is to run the application for a while, exercising all its features. Then find out how many relocatable memory blocks have been allocated. The HT command in MacsBug will provide this information. Once you have this number, divide by 64, add some slop (maybe 25%), and that's the number of times you need to call MoreMasters.

Don't use SetEventMask to disable mouseUp events. Better not to use it at all.
References: IM II-70, TN 202

SetEventMask should only be used to enable KeyUp events if an application needs them. Some programmers, especially in the early days, used SetEventMask to disable MouseUp events, which caused a lot of problems. One particularly strange one is that, after running them, double-clicking would no longer work in the Finder. Because the Finder could not see the MouseUp, it could not detect a double-click.

SetPort to a known good GrafPort once every time through the event loop.
References: IM I-165

You never know if a desk accessory is playing by the rules. Even if you do things absolutely correctly in your application, there is a bizarre set of circumstances which can (and has) resulted in crashes. Say desk accessory A plays hard and fast with the current GrafPort. Say it has the current GrafPort set to a window, and it closes the window. The current GrafPort becomes invalid. Now, desk accessory B does something which requires the current GrafPort to be valid, but does not explicitly set it first. This is common because there are calls that expect some GrafPort to be current, but don't really care which one, and some programmers are sloppy. When B tries this, kabloom! Even though your application is perfectly innocent, it will probably be blamed anyway. If you do a SetPort to a known good GrafPort every time through your event loop, this will not happen.

Calling WaitNextEvent with more than 50 ticks will fail on some systems.
References: TN 177

MultiFinder 1.0 had a bug which caused it to hang if you called WaitNextEvent from the background with more than about 50 ticks. The bug has since been fixed, but you never know what weird system will be used to run your application.

Set the cursor on suspend and resume events.

References: IM I-167, Programmer's Guide to MultiFinder

When you receive a resume event, another application may have changed the cursor, so set it to whatever you need it to be, even if it is just an arrow. When you receive a suspend event, set the cursor to an arrow as a courtesy to other applications whose programmers are not as well informed as you.

Call GetDblTime to get the maximum time for a double click.

References: IM I-261

The double-click time can be set by the user via the Control Panel. Use this value to determine whether two sequential clicks formed a double click. People have different capabilities and natural speeds, and some users may be physically unable to make a double click at a certain speed. Also remember that the value of GetDblTime can change at any time, because the user can pull down the Control Panel at any time.

Measure double-click time from mouse up to mouse down.

The Finder measures a double click from a MouseUp event to a MouseDown event. This seems to be a very natural way of doing it.

Call either WaitNextEvent or both GetNextEvent and SystemTask.

References: Programmer's Guide to Multifinder

WaitNextEvent is not just a direct replacement for GetNextEvent. It performs the function of SystemTask as well.

[Example Below is from Apple's TESample Source Code]

```
procedure EventLoop;

{Get events forever, and handle them by calling DoEvent.}
{ Also call AdjustCursor each time through the loop.}

       var
           cursorRgn: RgnHandle;
           gotEvent: BOOLEAN;
           event: EventRecord;
           mouse: Point;

   begin
       cursorRgn := NewRgn;             {we'll pass an empty region to WNE the first time
thru}
       repeat
           if gHasWaitNextEvent then
              begin  {put us 'asleep' forever under MultiFinder}
                  GetGlobalMouse(mouse);        {since we might go to sleep}
                  AdjustCursor(mouse, cursorRgn);
                  gotEvent := WaitNextEvent(everyEvent, event, GetSleep, cursorRgn);
              end
           else
```

```
    begin
        SystemTask;                    {must be called if using GetNextEvent}
        gotEvent := GetNextEvent(everyEvent, event);
    end;
if gotEvent then
    begin
```

```
                AdjustCursor(event.where, cursorRgn); {make sure we have the right cursor}
                DoEvent(event);                       {Handle the event only if for us.}
            end
         else
            DoIdle;
      {If you are using modeless dialogs that have editText items,}
      {you will want to call IsDialogEvent to give the caret a chance}
      {to blink, even if WNE/GNE returned FALSE. However, check FrontWindow}
      {for a non-NIL value before calling IsDialogEvent.}
      until AllDone
   end; {EventLoop}
```

Call IsDialogEvents and DialogSelect even if GetNextEvent returns false.

References: IM I-416

In addition to handling dialog events, these routines also control cursor flashing in dialog EditText items. They need to be called even as a result of null events to make sure the flashing occurs.

Use SetItem to include meta characters literally in menus.

References: IM I-346, IM I-357

AppendMenu recognizes certain characters as meta characters to control the text style, keyboard equivalents, and so on. If you want to display any of these meta characters within the text in the menu, you will need to use SetItem instead.

Never make MENU resources purgeable.

GetResource never produces resNotFound. Check for a NIL handle instead.

References: IM I-119

[quick example]

```
   id:=1000;
   myResHandle:=GetResource('ICON', id);
   if myResHandle<>nil then
      PlotIcon(r, myResHandle);
   else
      SysBeep(duration);
```

To create a resource file, call Create, then CreateResFile.

References: IM I-114, TN 101

CreateResFile will first check to see if the file exists, and if it does, it will add a resource fork to that file. CreateResFile uses the Poor Man's Search Path, so if you call it by itself, you may find that it actually modifies a file in the system folder rather than creating a file where you want it. To avoid this, call Create first to make sure a file exists where you want it.

To open a resource file read-only for shared access, use OpenRFPerm.

References: IM IV-17, TN 116, TN 185

Don't leave ResLoad set to false.

References: IM I-118

The SetResLoad routine should only be used in very special circumstances to prevent the automatic loading of resources into memory. Parts of the Toolbox require ResLoad to be true, so ResLoad should only be false for a very short time.

GetResource on a dctb may return a non-resource copy of the dctb.

When GetResource is called on a dctb, it may return a copy of the dctb rather than a handle to the resouce itself. This was done as a kludge to fix a bizarre problem. Of course, if you aren't aware of it, the kludge itself can cause some bizarre problems in programs that try to edit dctb resources.

Drag windows to the bounding box of GetGrayRgn.

References: IM I-282, IM I-289, IM V-208

DragWindow requires a bounds rectangle to specify the area over which the window can be dragged. Normally, it should be possible to drag a window anywhere. Once upon a time applications were recommended to use screenBits.bounds as the drag rectangle. Many pieces of Apple sample code recommend this to this day. However, on systems with multiple monitors, screenBits.bounds only enclose the screen with the menu bar. What you really want is to be able to drag the window within a rectangle that encloses all screens. The best one to use is the bounding box of the return value of GetGrayRgn. Because the GrayRgn covers all monitors, the bounding box is at least large enough. If the system is too old to have the GetGrayRgn function, you can use the GrayRgn global variable or just assume that screenBits.bounds is good enough. The following sample THINK C code drags theWindow around in response to an EventPtr called theEvent:

```
Rect boundsRect;
boundsRect = (*GetGrayRgn()) -> rgnBBox;
DragWindow(theWindow, theEvent -> where, &boundsRect);
```

Hide scroll bars when deactivating a window.

References: IM I-46

The entire scroll bar of an inactive window, including the arrows at the top and bottom, is supposed to be hidden. Few applications pay attention to this, but it is one of those little touches that really cleans up the feel of a program.

Call DrawGrowIcon when activating or deactivating a window with a grow region. DrawGrowIcon does not check to see if the window has a grow region. Call PenNormal before calling DrawGrowIcon.

References: IM I-287

DrawGrowIcon is not smart enough to know if the window has a grow region, so don't assume that you can always call it in a generic window handler. On the other hand, it is smart enough to know if the window is active or inactive, so it always does the right thing. It assumes certain things about the QuickDraw environment, such as a pen size of (1, 1), so it is safest to call PenNormal before every call to DrawGrowIcon.

itemHit will not be set when a dialog filter is called.

References: IM I-415, TN 112

This is one of the trickiest pitfalls to using dialogs. Dialog filters are nearly always used to provide interactions in userItems and are specific to those userItems. Thinking about that task, it is reasonable to assume that the filter can just test itemHit to see if the action is appropriate to the userItem. Though reasonable, the assumption is wrong. A dialog filter gets the raw event before any decisions are made. It is entirely responsible for determining where the click was and setting itemHit. An easy way to do this for a userItem is to do a PtInRect with the rectangle around the userItem. You can also use FindDItem.

[Sample below is from an old program that had a list manager in the dialog MXM].

```
function MyFilter (theDialog: DialogPtr; var theEvent: EventRecord; var itemHit: integer):
Boolean;

      var
         key: SignedByte;
         Itype: Integer;
         iBox, r: Rect;
         iHdl: Handle;
         MouseLoc: Point;
         isDbl: boolean;
         myList: ListHandle;
   begin
      Setport(theDialog);
```

```
MyFilter := False;
MyList := ListHandle(GetWRefCon(theDialog));
MyList^^.port^.txSize := 9;
MyList^^.port^.txfont := 4;
```

```
      case theEvent.what of
         keyDown, autoKey:
            begin
               key := theEvent.message;
               if key in [13, 3] then
                  begin
                     MyFilter := true;
                     itemHit := 9;          {Set Itemhit}
                  end;
            end;
         mouseDown:
            begin
               mouseLoc := theEvent.where;
               GlobalToLocal(mouseLoc);
               GetDItem(theDialog, 3, iType, ihdl, ibox);
               r := Ibox;
               r.right := r.right + 16;
               if PtInRect(mouseLoc, r) then
                  begin
                     isDbl := LClick(mouseLoc, theEvent.modifiers, Mylist);
                     MyFilter := true;
                     itemHit := 3;
                  end;
            end;
         UpdateEvt:
            begin
               BeginUpdate(theDialog);
               LUpdate(theDialog^.visrgn, Mylist);
               MyList^^.port^.txSize := 9;
               MyList^^.port^.txfont := 4;
               DrawDialog(theDialog);
               EndUpdate(theDialog);
            end;
      end;
   end;
```

Use a disabled UserItem to draw the roundrect outline around the OK button.

References: IM I-421, TN 34, default cdev

It seems strange that there is no trivial way to do such a basic feature of the Macintosh interface, but that's the truth. How does one make the roundrect outline around the default button that we all love so much? Just drawing it there after showing the dialog is not good enough, because if something happens to cover up the dialog, the outline will be erased. Some people use disabled PICT items containing roundrects, but you really have to make a custom PICT for every size of button to get the thickness of the line just right. A better way is to use a userItem that draws the outline. Add to the dialog a disabled userItem that is larger than the OK button by 4 pixels on each side. Make the dialog hidden, so that you will have to do a ShowWindow to show it. Then install a procedure for the userItem which draws the outline within its rectangle. Here are some THINK C code fragments:

```
pascal void DrawOKOutline(theDialog, whichItem)
DialogPtr theDialog;
INTEGER whichItem;
/*Draws an OK outline in a userItem*/
{
   INTEGER itemType;    /*The type of the item in GetDItem*/
   Handle theItem;  /*The handle to the item*/
   Rect itemBox; /*The box containing the item*/
   PenState savedPenState; /*The old pen state saved and later restored*/

   GetDItem(theDialog, whichItem, &itemType, &theItem, &itemBox);
   GetPenState(&savedPenState);
   PenNormal();
```

```
PenSize(3, 3);
FrameRoundRect(&itemBox, 16, 16);
SetPenState(&savedPenState);
```

```
}
   ...and when you want do use this userItem...

#define MyDlgRSRC whatever /*Number of the dialog resource*/
#define MyDlgOutline whatever    /*The number of the userItem with the outline*/

   INTEGER itemType;    /*The type of the item in GetDItem*/
   Handle theItem;  /*The handle to the item*/
   Rect itemBox; /*The box containing the item*/

   /*Get the dialog box*/
   theDialog = GetNewDialog(MyDlgRSRC, (DialogPtr) 0, (WindowPtr) -1);

   /*Add the OK button outline*/
   GetDItem(theDialog, MyDlgOutline, &itemType, &theItem, &itemBox);
   SetDItem(theDialog, MyDlgOutline, itemType, &DrawOKOutline, &itemBox);

   /*Everything is set; show the dialog*/
   ShowWindow(theDialog);
```

If you want to get really fancy you can have the userItem procedure first look at the rectangle of the OK button, and then set its own rectangle accordingly.

[For an automatic way of doing this look for the Default cdef article by Lloyd Lim]

ModalDialog assumes the dialog is already visible and in the front.

References: IM I-415

A dialog has to be visible and above all other windows before you call ModalDialog. If you call ModalDialog without ensuring this, there will be big trouble.

Use screenBits.bounds to center dialogs, alerts, etc. below the menu bar.

References: IM I-163, IM I-527

screenBits.bounds is the bounding rectangle of the screen that has the menu bar. This is the most convenient place to put dialogs and alerts. With some simple calculation, knowing screenBits . bounds and the width and height of your window, you can center it. You can find out the width and height of the standard file dialogs by examining their DLOG templates.

If you save window locations in files, they may not be valid for all monitors.

Refernces: IM V-208

Saving the locations of windows between sessions is a nice thing for an application to do, but what if the user opens the document on a system with a very different screen setup? The window might not appear where it can be manipulated. So, if you do this, be sure to test to see if a window will appear in an accessible place on the screen, and if not, put it in a default location. You can do this by checking to see if a rectangle inset a few pixels within the title bar intersects the grayRgn.

DragWindow expects startPt in boundsRect; if not it may not call SelectWindow.

References: IM I-289

If the startPt you pass to DragWindow is not within the boundsRect, a click in the title bar may not call SelectWindow and bring the window to the front. Dragging will still work, but clicking in place won't. The easiest fix is just to make sure boundsRect is big enough. You can call SelectWindow yourself, but this affects the feel of window dragging.

SelectWindow does not automatically call SetPort. You must do that yourself.

References: IM I-284

Selection and drawing are independent of each other, although most of the time you want to do both. You can draw into windows that are not selected. QuickDraw does not set the port behind your back, which has the advantage of giving you flexibility. It has the disadvantage that you must remember to set the port yourself.

DialogSelect responds to activate events but ignores suspend/resume events.

References: IM I-417

If you get suspend and resume events, make sure to activate or deactivate dialogs on the screen in response to them.

Call PenNormal before calling DrawControls.

References: IM I-322

DrawControls assumes several things about the QuickDraw environment. Calling PenNormal is the best way to be sure that it does not mess up.

The Control Manager only works when the origin of a window is at (0, 0)

References: IM I-322

If you use SetOrigin to set the origin of a window, you must call SetOrigin(0, 0) before doing anything that can affect or draw controls.

To find the position of a window, use LocalToGlobal on the origin.

References: IM I-165, IM I-193

The safest way to find the position of a window is to do a SetPort to the window, get the origin of the window, and do a LocalToGlobal on that point.

Always set the VisRgn and ClipRgn of offscreen ports.

References: IM I-149

The assertion in Inside Macintosh that the VisRgn has no effect on images that are not on the screen is wrong. To be safe, always set both regions of offscreen GrafPorts.

Set the ClipRgn first when making a picture.

References: TN 59

The default ClipRgn for pictures is the largest possible region. If you create a picture using the default, as soon as you try to scale or move it using DrawPicture, the edges can overflow, causing all sorts of nasty things to happen. To avoid this, set the ClipRgn as the first thing when you create a picture.

Don't make rowBytes in bitMaps greater than 8191.

References: IM V-53

The top three bits of rowBytes are used by the first release of Color QuickDraw to keep information about the bitmap. More recent versions of Color Quickdraw only use the top two bits, but do you want to gamble that everybody in the world has upgraded? I thought not.

To dim text, draw a rectangle with penPat=gray and penMode=patBic over it.

Dim text is another of those things which is strangely absent from QuickDraw. You can dim text or anything else by just painting over it with a gray penPat in patBic mode.

To draw rotated text, draw to an offscreen bitmap, rotate it, and CopyBits it.

Although there are ways to rotate text at any angle in PostScript, QuickDraw is only able to draw text in one orientation. The only way to get rotated onscreen text is to rotate the bitmap. There is no call to do this; you have to do it yourself.

Don't use picSize to determine the size of a picture. Check the handle size.

References: IM V-87

The picSize field of a picture is only 16 bits wide. This is O.K. for most black-and-white pictures, but when Color QuickDraw was invented, it was discovered that this limitation was way too small. The actual length of a picture is now determined by the size of the handle. Although picSize is maintained for compatibility in version 1 pictures, don't count on it.

Never draw outside a window except in XOR mode for temporary effects like drag.

Drawing outside a window is very dangerous. For one thing, historically, most programs which have done this broke when extensions were made to QuickDraw. For another thing, in most cases, the user does not expect anything except the Finder to draw on the desktop. If you need to draw in a window the size of the screen, why not open a window the size of the screen and use that? In general, only DragGrayRgn and XOR lines for zooms should be done outside a window. These are temporary effects which always go away, so they are not so bad.

To avoid animation flicker, synchronize drawing to the vertical retrace.

References: IM II-350, IM V-567

The video hardware is constantly sweeping through the screen memory displaying the bytes on the screen. If you try to access some memory during a QuickDraw operation that is currently being swept, there will be a conflict. Usually, QuickDraw is so fast that you will not notice it, but if you are doing animation using CopyBits, you might notice some flicker. You can try to avoid this by synchronizing the beginning of the CopyBits to the vertical retrace. The idea is always to have the drawing occur just ahead of the sweep so that no conflict occurs. A good way of finding out when the vertical retrace occurs is to install a VBL task on a small Macintosh, or use SlotVInstall on a Macintosh with slots. The routine will be called as a result of the vertical retrace, so it can signal another portion of the program to begin drawing. You will have to experiment with timing for your particular application.

Lock handles to pictures before calling DrawPicture.

References: IM I-190

Rumor has it there is a bug in DrawPicture which involves following a pointer to the picture data rather than a handle and an offset. As DrawPicture can at times cause a memory compaction, this is unsafe. I find this bug hard to believe, as it would be a monstrous oversight, but better safe than sorry. Lock the picture.

CopyBits on more than 3Kb data will work, but it might have to allocate memory.

References: IM I-188

On old systems, CopyBits had a problem with copying a picture bigger than about 3Kb. This was because CopyBits uses temporary memory from the stack. The bug required some programs to break down CopyBits calls into a number of thin wide strips, which was not very convenient. The bug has for a long time been fixed. What happens now is that CopyBits will first try to get enough memory from the stack and, if that is not enough, will allocate blocks using Memory Manager calls.

The small Mac screen is 512 pixels wide by 342 pixels high including menu bar.

Count 'em.

Don't call any Memory Manager routines during an interrupt.
Unlocked handles may not be valid during an interrupt.

References: IM II-195

An interrupt may interrupt anything, including a memory compaction. During a memory compaction, parts of the heap may be in an indeterminate state. Calling any Memory Manager routines could destroy the heap. Also, it is unsafe to look at unlocked handles. The memory in the handle may have been in the process of moving when the interrupt occurred.

To synchronize to the vertical retrace on Macs with slots, use SlotVInstall.

References: IM V-567

Don't write in the application file. This will fail with read-only devices.
References: TN-115,TN-116

You never know where the file containing your application resides. It could be on a CD-ROM, or it could be on a shared volume. Applications that write to themselves, for example to set user preferences, will fail under these circumstances. Besides, there is a better way to save preferences, described in another one-liner.

Use PBGetVInfo to convert a VRefNum to a volume name.
References: IM IV-129

Delete uses the Poor Man's Search Path, so don't delete blindly.
References: IM IV-113, IM IV-147

As the high level FSDelete is an old MFS call, it will use the Poor Man's Search Path. This means that, if it does not find a file of the right name in the current directory to delete, it will try to delete such a file in the System Folder. This can be disastrous. Before you call Delete, make absolutely sure that it will delete the correct file by checking to see if such a file exists where you expect it.

File Manager routines with dirID=0 use the Poor Man's Search Path.

Actually, every File Manager routine with dirID=0 or no dirID specified uses the Poor Man's Search Path. Most of the time this just makes things more convenient, as it allows the System File to be searched by default after the current directory. Beware of what is happening, though.

Truncate and reallocate files before overwriting to reduce fragmentation.
References: IM IV-111, IM IV-112

As you overwrite a file again and again with sometimes less and sometimes more data, the file can become fragmented. Lots of fragmented files will worsen the performance of the entire file system. To help avoid this fragmentation, every time you need to overwrite a file from the beginning to the end, first truncate it to zero bytes using the SetEOF function and then allocate the number of bytes you expect to write. The FileManager will try to allocate a nice contiguous chunk of blocks.

Check/change the creator and type of Save As... files before overwriting.
References: IM IV-113

Just checking for the existence of a file with the same name is not enough. You must make sure that the file type and creator are correct as well. Otherwise, the file will not appear correct to the Finder, and you may not be able to read it in the future. There are a variety of strategies to do this. Some programs simply don't let you Save As over a file of a different type or creator. Some quietly change the type and creator. Some put up an alert. You decide.

If you rewrite files by deleting and creating, copy all Finder information.
References: IM IV-113

Deleting and recreating is not the best way to rewrite a file, but if you must do it, be sure to copy all the Finder information. This means the entire contents of the FInfo record. One more piece of information that needs to be copied is the GetInfo comment. I don't think there is a standard way of doing this, which is one of the reasons deleting and recreating is a bad thing to do.

Directory ID's are longs, not shorts. Shorts work ALMOST all the time.
References: IM IV-92

If a file version number appears in a file manager call, always set it to 0.
References: IM IV-90

The version number of a file was an old mechanism to distinguish between two files with the same name, for example to use one as a backup for the other. It was a good idea, but it was never completely implemented. The problem is that different bits of software do different things with the version number. The Finder ignores the

version number, but the standard file routines only show files with version 0. This is the most common cause of bugs involving a file that you can see in the Finder but not within an application. To prevent this from happening, if a version number appears in a call, always explicitly set it to 0 before you make that call.

To convert a pathRefNum to a name or file number, use PBGetFCBInfo.

References: IM IV-179

Prevent the creation of files with names that begin with a period.

References: IM II-175, IM II-245

The File Manager interprets any file name beginning with a period as the name of a device driver. Unfortunately, it is also possible to create real files with names that begin with a period, causing no end of confusion. To protect your users from this, add a check to your Save As routines to reject file names that begin with a period.

Write/update the Finder info before writing data or resource forks.

References: IM IV-113

Lock handles before passing their dereferenced pointers to any routine.

References: IM XREF

It is well known that some Toolbox routines are known to change memory. The Inside Macintosh X-Ref has a list of such routines, which it claims is complete. Hah! The list is really growing all the time, and you cannot count on any routine's being safe any more. Even if you could, you run the risk that your language accesses that routine through glue which can be loaded the first time you call the routine, thus changing memory. The best way to be safe from this is to lock all handles before dereferencing their pointers and passing them to any routine, however benign the routine may seem.

Lock handles before setting referenced data to expressions containing functions

This is a nasty one. Let us say you have a C statement like `(*aHandle)[5] = foo(3);` in other words, aHandle is a handle to an array, and you want to set element number 5 of that array to the return value of foo. This is unsafe! The reason is that C will calculate the pointer to the fifth element of aHandle before it calls foo. If foo compacts memory, this pointer will no longer be valid. This problem is not limited to C; it can occur with any language that allows similar constructs. Don't count on a certain order of evaluation. Lock the handle, or use an intermediate variable.

Put an odd long at location zero on a 68000 to help find NIL handle references.

Whenever you use a NIL handle or pointer, it will look at whatever is stored in location 0. The long word at location 0 could conceivably point anywhere, causing all sorts of indirect problems to result from nil handle references. If you suspect your application has NIL handle references, you can track them down by using a Macintosh that uses the old 68000 processor. If you put an odd value at location 0, whenever your application tries to use a NIL handle, you will get an exception immediately, and you will break into the debugger.

Call MoveHHi before locking a handle to avoid memory fragmentation.

References: IM II-44

If you lock a handle, it may be in the middle of the heap. If you lock several handles, they may fragment memory and affect subsequent memory allocations. You can avoid this by first calling MoveHHi on the handle before locking it. MoveHHi will take some time, but it will try to move the handle as high in memory as possible before you lock it.

Use HGetState/HLock/HSetState to lock a handle then put it back as it was.

References: IM IV-79

Most of the time, the best way of using handles is always to keep them unlocked and only lock them when it is absolutely necessary for a short time. However, sometimes you have a handle that you don't know is locked and you need to lock it and then set it back to the previous state. To do this, use HGetState to get the state of the handle before locking it. Then, instead of using HUnlock, use HSetState to restore the handle to its previous state.

Always use unsigned characters within text and Pascal-format strings.

The first character of a Pascal format string is its length in bytes, from 0 to 255. This length only makes sense if you are using unsigned characters. If you are using signed characters, such as the default C char, numbers above 127 will be interpreted as negative numbers. This is very dangerous, especially when you use packages such as TextEdit, which take a length. Most languages will quietly promote a signed character to a signed short or long and will happily pass this value to TextEdit, which will interpret it as a VERY LARGE length, whereupon TextEdit will crash and burn. To avoid this, always use unsigned characters.

Save application preferences in a folder named Preferences in the System Folder

As it is a bad idea to keep application preferences in the file itself, where do you keep them? After a discussion of this matter on Usenet, the consensus seemed to be that there should be a folder named Preferences in the System Folder and each application should have file with an application-specific name in that folder. To make it as general as possible, the algorithm should work like this: First look in the current directory for the preferences file. If it is found, use that. The Poor Man's Search Path will ensure that any file in the System Folder will be found as well. If no file is found, look for a Preferences folder in the System Folder and look for the file there. If none is found, search all folders in the System Folder for the file. (This allows the user to rename the Preferences folder.) If it is not found, create a Preferences folder and save a file with the default preferences in that folder.

Use SysEnvirons to find the System (Blessed) Folder.

References: IM V-5

Use GetAppParms to get the name of the application.

References: IM II-58

The high bit of SysParam . volClik enables the alarm clock.

References: IM II-370

Check the application name at $910 before exiting with ES within MacsBug.

Hitting the programmer's interrupt switch and doing an ES within MacsBug is a good way to stop a runaway application. Unfortunately, with MultiFinder, you never know just what is running when you hit that switch. To find out, look at location $910 to see the name of the current application. If it is not the one you want to stop, enter G and try again.

To exit to shell in the mini-debugger, enter SM 0 A9 F4 and then G 0.

The mini-debugger has very few commands. One of the most useful functions, exit to shell, is not provided. There are a lot of ways of doing an exit to shell, but most of them involve remembering magic numbers of addresses for different systems. The way presented here should work on any machine. A9F4 is the code for the ExitToShell trap. SM 0 A9 F4 puts that instruction in the RAM at location 0. G 0 jumps to that instruction and executes it.

In Pascal, don't nest procedures to be passed by procedure pointer.

In Pascal, "with theHandle^^" is unsafe if memory compaction can occur.

Because a with statement dereferences a handle at the beginning of the block of code it controls, if the code causes a memory compaction, this handle may no longer be valid. Lock the handle first.

Avoid writing tail patches for traps.

References: TN 212

Some Apple patches to traps check the stack to see who called them and have some special cases. I know that this is not very sociable of them, but the upshot is that your application will get blamed if you use a tail patch and this causes one of Apple's bug fixes to fail.

There is no official way to tell if MultiFinder is running or not.

I wish there were a way to tell this, because it affects the behavior of Launch, but there isn't. There have been a variety of unofficial ways of telling, but most of them have failed as the system was changed. The

only absolutely safe way to tell seems to be to assume that you are not running under MultiFinder until you receive a MultiFinder event.

Scheme to Manage a "Windows" menu

by Ben Cranston

Back in August of 1989 there was a discussion here on methods for implementing a "Windows" menu, one with an entry for each window currently displayed by the application. The act of selection would bring that window to the fore, etc.

David Phillip Oster asked: "what happens if you have multiple files open, all with the same name?" and then suggested "perhaps the simplest solution is to just use the item position in the windows menu, rather than the title, to determine which window the user wishes to select -- a problem with this is that you can get identical menu items, but at least the program can distinguish among them, if not the user."

This posting describes a simple scheme to implement this paradigm. Please feel free to use your "junk" key if you are not interested.

The scheme comprises two parts:

1. A linked list of the windows, sorted by the window names, is kept from a global cell through a window pointer variable in each window data area. Note: this list is in addition to the Window Manager's list.

2. A small integer in the window data area holds the index in the menu of that window's item. I called it MRefNum ("Menu Reference Number), for lack of a better name.

The management algorithms are:

When a new window is created, the proper place in the linked list must be found. We scan the linked list until we find either the end of the list or a window whose name is lexically greater than that of our new window. We are then interested in the item BEFORE that position which is either the root or a window whose name is lexically less than the new name.

The MRefNum of this preceding entry determines the position in the menu that the new window's name is inserted. It is also incremented and becomes the MRefNum of the new window. The new window is inserted into the chain at this point, and all succeeding windows in the chain have their MRefNum's incremented, corresponding to the fact that their menu entries were pushed down by the insert of the new window name.

If the previous item was the root, the new MRefNum becomes 1, and the same things are done.

When a window is destroyed, the menu item corresponding to its MRefNum is deleted, the window is removed from the chain, and all succeeding windows in the chain have their MRefNum's decremented.

I did have an algorithm for changing the name, which broke down into two possibilities corresponding to moving a name UP in the chain (incrementing the MRefNums between the new and old position) or moving a name DOWN in the chain (decrementing the MRefNums between the old and new position) but in the actual program I deleted and re-inserted the window :-)

Getting from a window to it's menu item is easy, as the MRefNum designates the appropriate menu index. Getting from a menu item to the window is just searching for the appropriate MRefNum value.

Here's the code for creating a window:

```
/* Insert a window into the proper point in the window chain.
* Also makes appropriate entry in the window menu.
```

```
*/
```

```
InsertWind(RWPtr newwind)
{
   RWPtr   lastw = nil;
   RWPtr   nextw = AFWind;  /* the root */
   short   refn;

   while ( (nil!=nextw) && (0<CompHand(newwind->wname,nextw->wname)) ) {
    lastw = nextw;
    nextw = nextw->next;
   }

   refn = (nil!=lastw)?lastw->mrefn:0;
   newwind->mrefn = refn+1;
   newwind->next = nextw;

   if (lastw!=nil)
    lastw->next = newwind;
   else
    AFWind = newwind;

   InsMenuItem(WMenu,"\pa",WMBASE+refn);
   SetItem(WMenu,WMBASE+newwind->mrefn,*newwind->wname);

   while (nil != nextw) {
    nextw->mrefn++;
    nextw = nextw->next;
   }

}

Here's the code for destroying the window:

/* This routine removes a window from the window chain.
* It also removes the appropriate window menu entry.
*/

RemoveWind(RWPtr mortwind)
{
   RWPtr   lastw = nil;
   RWPtr   nextw = AFWind;  /* the root */

   while ( (nil!=nextw) && (nextw!=mortwind) ) {
    lastw = nextw;
    nextw = nextw->next;
   }

   if (nil!=nextw) {
    if (nil!=lastw)
        lastw->next = nextw->next;
    else
        AFWind = nextw->next;
    DelMenuItem(WMenu,WMBASE+mortwind->mrefn);
    while (nil!=nextw) {
        nextw->mrefn--;
        nextw = nextw->next;
    }
   } else
    SysBeep(30);
}
```

Here's some code (case WIMENU) that goes from the menu item to the window.

```
/* Process selection from menu.
* Apple menu:
*   Note: MUST do DA stuff so MF can switch from Apple menu...
```

```
*  Note: no "about" box is implemented.
* FILE menu:
*  ...
* SHOW menu:
*  Flip view bits or change displayed resource type.
* WIND menu:
*  Iconize/deiconize or bring window to front.
*/

DoMenu(int menuparam)
{
   char       daname[256];
   int        menuitem = LoWord(menuparam);
   WindowPtr  windp = FrontWindow();
   RWPtr      mywind = windp;
   short      wkind = WindowKind(windp);

   switch ( HiWord(menuparam) ) {
   case APMENU:
   if ( menuitem == 1 )
       SysBeep(30);
   else {
       GetItem(AMenu,menuitem,daname);
       OpenDeskAcc(daname);
       AdjustMenus();
   }
   break;
   case EDMENU:
   SystemEdit(menuitem-1);
   break;
   case FIMENU:
   switch (menuitem) {
   case FMOPEN:
       OpenWindow();
       break;
   case FMCLOS:
       if (wkind < 0)
      CloseDeskAcc(wkind);
       else
      KillWindow(windp);
       break;
   case FMPSET:
       PrintSetup(windp);
       break;
   case FMPRNT:
       PrintWindow(windp);
       break;
   case FMQUIT:
       ExitToShell();
   }
   break;
   case SHMENU:
   if (nil != windp) {
       if (menuitem >= SMBASE) {
      OpenType = menuitem-SMBASE;
      SetWindType(windp,OpenType);
       } else switch (menuitem) {
       case SMRNUM:
      mywind->wview.show.rnum = mywind->wview.show.rnum ^ 1;
      break;
       case SMRNAME:
      mywind->wview.show.rname = mywind->wview.show.rname ^ 1;
      break;
```

```
 case SMMASK:
mywind->wview.show.mask = mywind->wview.show.mask ^ 1;
 }
```

```
        PostWindowSize(mywind);
        SetPort(windp);
        InvalRect(&windp->portRect);
        AdjustMenus();
    }
    break;
    case WIMENU:
    if (menuitem<WMBASE) {
        if (nil != windp) {
        switch (menuitem) {
        case WMICZE:
            if (mywind->wview.show.icize)
           DeIconizeWindow(mywind);
            else
           IconizeWindow(mywind);
        }
        SetPort(windp);
        InvalRect(&windp->portRect);
        AdjustMenus();
        }
    } else {
        for ( mywind=AFWind ; mywind!=nil ; mywind=mywind->next)
        if (WMBASE+mywind->mrefn == menuitem)
            SelectWindow( (WindowPtr) mywind);
    }

    }
    HiliteMenu(0);
}
```

Here's some code (last for loop in this proc) that gets from window to menu:

```
/* Set checkmarks of view and window menus according to mode of front window.
* There are three cases:
* If FrontWindow() is nil then there are no windows open (degenerate case).
* If FrontWindow() is not nil but windowKind is negative then the window is
*  a Desk Accessory opened by (single) Finder or a DA opened by MultiFinder
*  within our application heap (option launch).  In this case we want to
*  make the EDIT menu active but not any of our own menus, since the window
*  will not have the data items our code assumes are there.
* If windowKind is positive we assume it is one of our own normal windows,
*  since we do not have any modeless dialogs.
*/

AdjustMenus()
{
    WindowPtr  windp = FrontWindow();
    RWPtr      mywind = windp;
    short      indx;

    if (nil == windp) {
    DisableItem(FMenu,FMCLOS);
    DisableItem(FMenu,FMPSET);
    DisableItem(FMenu,FMPRNT);
    DisableItem(EMenu,0);
    DisableItem(SMenu,0);
    CheckItem(SMenu,SMRNUM,false);
    CheckItem(SMenu,SMRNAME,false);
    CheckItem(SMenu,SMMASK,false);
    for (indx=0 ; indx<NRType ; indx++)
        CheckItem(SMenu,SMBASE+indx,false);
    DisableItem(WMenu,0);
```

```
CheckItem(WMenu,WMICZE,false);
} else if ( WindowKind(windp) < 0 ) {
EnableItem(FMenu,FMCLOS);
DisableItem(FMenu,FMPSET);
```

```
    DisableItem(FMenu,FMPRNT);
    EnableItem(EMenu,0);
    DisableItem(SMenu,0);
    CheckItem(SMenu,SMRNUM,false);
    CheckItem(SMenu,SMRNAME,false);
    CheckItem(SMenu,SMMASK,false);
    for (indx=0 ; indx<NRType ; indx++)
        CheckItem(SMenu,SMBASE+indx,false);
    DisableItem(WMenu,0);
    CheckItem(WMenu,WMICZE,false);
    } else {
    EnableItem(FMenu,FMCLOS);
    EnableItem(FMenu,FMPSET);
    EnableItem(FMenu,FMPRNT);
    DisableItem(EMenu,0);
    EnableItem(SMenu,0);
    CheckItem(SMenu,SMRNUM,mywind->wview.show.rnum);
    CheckItem(SMenu,SMRNAME,mywind->wview.show.rname);
    CheckItem(SMenu,SMMASK,mywind->wview.show.mask);
    for (indx=0 ; indx<NRType ; indx++)
        CheckItem(SMenu,SMBASE+indx,indx==mywind->wdata.dtype);
    EnableItem(WMenu,0);
    CheckItem(WMenu,WMICZE,mywind->wview.show.icize);
    }

    for ( mywind=AFWind ; mywind!=nil ; mywind=mywind->next)
    CheckItem(WMenu,WMBASE+mywind->mrefn,( (RWPtr) windp == mywind ));

    DrawMenuBar();
}
```

These are subroutines used by the above:

```
/* Compare string handles
*/

CompHand(Handle s1,Handle s2)
{
    int ans;

    HLock(s1);
    HLock(s2);
    ans = IUCompString(*s1,*s2);
    HUnlock(s1);
    HUnlock(s2);
    return(ans);
}

/* Get the windowKind field from the window record.  This is used to
* decide if a window belongs to the program, or if it is a desk
* accessory called either from the old Finder environment or from
* the MultiFinder environment with the option key down.
*/

WindowKind(WindowPtr windp)
{
    int wkind = 0;

    if (nil != windp)
    wkind = ((WindowPeek) windp)->windowKind;
    return(wkind);
}
```

Actually, it occurs to me that there is no need to explicitly store the MRefNum at all, as it should be identical to the position of that window in the list (that is, reading the list should return 1, 2, 3, etc). Extension to remove the MRefNum cell is left as an exercise to the reader...

How to write an INIT in Pascal

by Matthew Xavier Mora

This started out to be an "is it possible to write an INIT in PASCAL?" project to see if I could do It. The answer is yes and no. It can be done but you will need either some inline assembly or some assembly glue code. The INIT I wrote is a jGNEfilter that checks to see if the user hit one of the extended keyboard function keys. The unit "newJGNEFilter" is not real working code. It is only a framework of what you need to do. I myself have not finished the code yet so I am not sure if it will be the way I go. When my INIT is finished I will update this article to include full working code.

```
{This unit is the code that will install a jGNEFilter patch.}
{Written by Matthew Xavier Mora}

unit install;
interface
   procedure main;

implementation
   procedure ShowINIT (iconID, moveX: Integer);
   EXTERNAL;
   procedure main;
      var
         newjgne: Handle;
         addr: longint;
         JGNE: ptr;
         jgneptr: ^ptr;
   begin
      SetZone(SystemZone);
      newjgne := Get1Resource('CODE', 128);
      if newjgne <> nil then
         begin
            DetachResource(newjgne);
            HLock(newjgne);
            JGNE := pointer($29A);
            BlockMove(JGNE,ptr(ord(newjgne^)+10),4);{move jgne address into header of
newcode}
            jgneptr := pointer($29A);
            jgneptr^ := pointer(newjgne^);
            ShowINIT(128, -1);
         end;
   end;
end.

unit newJGNEFilter;

interface

   procedure main;

implementation
   function GetA1 (dummy: longint): longint;
   external;
   function GetD0 (dummy: longint): integer;
   external;
   procedure Setresult (dummy: integer);
   external;
   procedure DoJsr (addr: ProcPtr);
   inline
      $205F, $4E90;
```

```
procedure DoJmp (addr: ProcPtr);
inline
```

```
    $4CDF, $1CE0, $4E5E, $205F, $4ED0;
           {MOVEM.L    (A7)+,D5-D7/A2-A4}
           {UNLK       A6}
           {MOVEA.L    (A7)+,A0}
           {JMP        (A0)}


procedure main;

    var

       addr, oldjgne: procptr;
       dummy: integer;
       Eventmessage: longint;
       event: eventrecord;
       hasevent: boolean;
       Evntptr: ^eventrecord;
begin
    hasevent := Boolean(GetD0(0));      {D0 contains flag of gne}
    if hasevent then
       begin
          Evntptr := pointer(geta1(0));   {A0 contains a pointer to the eventrecord}
          Eventmessage := Evntptr^.message;
          {do whatever you wish }
          {if you hande the event your self then set D0 to false}
          { SetD0(false);}
       end;
    oldjgne := procptr(ord(@main) - 6);     {get address that was stored into header}
    addr := procptr(oldjgne);
    if addr <> nil then
       DoJmp(addr);                      {jmp to address stored in header}
end;
```

How do you play Asynchronous Sound?

by Larry Rosenstein

[Asynchronous Sound Code]

Enclosed is the source for an MPW Pascal unit that shows how to play asynchronous sounds with the Sound Manager. I have tried this unit only on System 6.0.2; supposedly there are bugs in earlier versions of the Sound Manager. This unit also doesn't check for the existence of the Sound Manager, I assume that you do this at a higher level.

I used this in a simple MacApp program that will open any file and allow you to play any snd resource in the file ansynchronously. (I started this with the idea of allowing copy & paste, but haven't gotten that far yet. If there is interest, I can post that program.)

Larry Rosenstein, Object Specialist

```
---
(*
A simple unit that demonstrates how to produce asynchronous sound with
the Macintosh Sound Manager.  Although I am pretty confident about this code,
I don't guarantee that this code demonstrates the correct way to do things.  It
does seem to work reliably.

This unit doesn't solve any of the tricky issues about using the Sound Manager.
Primarily, sound channels should be disposed of as soon as they are no longer
needed.  This code does just that, but it doesn't prevent your program or a
background program from trying to make sound.

Changes:

2/16/89  Lock the sound resource; state restored in call back
gSoundPlaying is the actual handle.

Larry Rosenstein
Apple Computer, Inc.
lsr@Apple.COM

Copyright 1988-1989 Apple Computer Inc.  All Rights Reserved.
*)

UNIT UAsynchSnd;

INTERFACE

USES
MemTypes, Quickdraw, OSIntf, ToolIntf;

{ call this before any other routine }
PROCEDURE InitUAsynchSnd;

{ returns TRUE if an asynchronous sound is playing }
FUNCTION  IsSoundPlaying: BOOLEAN;

{ equivalent to SndPlay, but does it asynchronously; if you call this
while another sound is playing, the first one will be stopped }
FUNCTION  ASynchSndPlay(sndHandle: Handle): OSErr;

{ stop the sound from playing; may be called even if no sound is
currently playing }
```

```
PROCEDURE StopAsynchSound;
```

```
{ should be called when your program exits }
PROCEDURE TerminateUAsynchSnd;

IMPLEMENTATION

VAR
gSoundPlaying:          Handle;
gSoundState:            SignedByte;
gSndChannel:            SndChannelPtr;

PROCEDURE ChanCallBack(chan: SndChannelPtr; cmd: SndCommand); FORWARD;
FUNCTION  GetA5: LONGINT; INLINE $2E8D; {MOVE.L A5,(A7)}
FUNCTION  LoadA5(newA5: LONGINT): LONGINT;
                                        INLINE $2F4D,$0004,$2A5F;

(********************)

PROCEDURE InitUAsynchSnd;
BEGIN
gSndChannel := NIL;
gSoundPlaying := NIL;
END;

FUNCTION  ASynchSndPlay(sndHandle: Handle): OSErr;
VAR     err:                    OSErr;
aCommand:               SndCommand;

BEGIN
StopAsynchSound;        { kill the current sound & channel }

err := noErr; { default value }

{ gSndChannel should be NIL now }
err := SndNewChannel(gSndChannel, 0, 0, @ChanCallBack);
{ We don't specify a synthesizer, since we are assuming that
the snd resource specifies one.  For example, the
standard Clink-Klank snd resource doesn't use the
sampled synthesizer. }

gSoundState := HGetState(sndHandle);
MoveHHi(sndHandle);
HLock(sndHandle);
gSoundPlaying := sndHandle;

IF err = noErr THEN
err := SndPlay(gSndChannel, sndHandle, TRUE);

WITH aCommand DO BEGIN
cmd := callBackCmd;
param1 := 0;
param2 := GetA5;
END;
IF err = noErr THEN
err := SndDoCommand(gSndChannel, aCommand, FALSE);

IF err <> noErr THEN
StopAsynchSound;                { flush channel; unlock sound }

AsynchSndPlay := err;
END;

PROCEDURE ChanCallBack(chan: SndChannelPtr; cmd: SndCommand);
VAR     oldA5:  LONGINT;
```

```
BEGIN
oldA5 := LoadA5(cmd.param2);    { get the application's A5 and set it }
```

```
HSetState(gSoundPlaying, gSoundState);
gSoundPlaying := NIL;

oldA5 := LoadA5(oldA5);                 { restore old A5 }
END;

FUNCTION  IsSoundPlaying: BOOLEAN;
BEGIN
IsSoundPlaying := gSoundPlaying <> NIL;
END;

PROCEDURE StopAsynchSound;
BEGIN
IF gSndChannel <> NIL THEN BEGIN
IF SndDisposeChannel(gSndChannel, TRUE) = noErr THEN { nothing };
gSndChannel := NIL;
END;
IF gSoundPlaying <> NIL THEN
BEGIN
HSetState(gSoundPlaying, gSoundState);
gSoundPlaying := NIL;
END;
END;

PROCEDURE TerminateUAsynchSnd;
BEGIN
StopAsynchSound;
END;

END.
```

Default 2.1 CDEF

By Lloyd Lim

The Default CDEF is a simple aid for Macintosh programmers that draws default button outlines for any size buttons, in the proper color, in your application and in ResEdit dialog and alert templates. Push buttons, check boxes, and radio buttons can also be drawn using the window's font.

Instructions

Default is a control definition function that you can copy and paste into your application. The Default CDEF enhances the System file's standard control definition 0. If the last character of a button's title is an '@' (an at or an each symbol), the button is drawn with an outline indicating that it is the default button. The trailing '@' is not drawn with the title.

If the button is inactive, the outline is grayed out. When Color QuickDraw is available, outlines are drawn in the same color as the button's frame. If the Default CDEF is in the same file as your application's DITL resources, ResEdit will display default button outlines drawn by Default. You do not need to write any code to use Default.

A dialog's default button can be changed at any time in your application simply by changing the appropriate button titles. However, you, the programmer, are responsible for making sure there is only one default button. If you are not using a filterProc, Default makes sure that pressing the Return key or Enter key is the same as clicking in the default button. The default button is also highlighted when the Return key or Enter key is pressed. If you are using your own filterProc, you must perform these tasks.

If the last character of a control's title is an '*f*', the control is drawn using the window's current font. Push buttons, check boxes, and radio buttons will look the same as if you used a CNTL resource with the useWFont variation code. This feature is especially useful for CDEVs. If you need a default button that uses the window's font, end the button's title with '*f*@'.

Compatibility

Default should work on any model of Macintosh with any System version. Default is 32-bit clean. Except for drawing the outline, Default lets the System file's CDEF 0 do practically everything. Thus, Default's buttons do anything that normal buttons do.

Be warned that Default uses the contrlData field of a ControlRecord (which is okay since it is reserved for CDEFs). If your application does something strange with this field, Default will not work.

When you drag or resize a default button in ResEdit, you may notice some slight flickering. This occurs because ResEdit does not know about the outline so Default forces ResEdit to refresh portions of the window. Default only behaves this way in ResEdit. This does not occur in normal applications. Default now comes in versions with and without the ResEdit updating code. This may be helpful if you are concerned about your application's size.

Bugs

The outline will not update if the outline needs updating and the button does not, and UpdtControl or UpdtDialog is being used to update the controls. This problem does not occur with normal modal dialogs and occurs rarely with modeless dialogs. Unfortunately, there doesn't seem to be a clean solution this problem. If you have this problem, you can call Draw1Control for the default button on every update event or simply use DrawControls or DrawDialog instead.

If you have a default button and you aren't using a filterProc, pressing the Return key or Enter key will click in the default button even if the button is hidden. This is a bug in the Dialog Manager. If you have this problem, you must change the default button to a normal button before you call HideDItem or HideControl.

Please report any other bugs to any of the addresses listed below. Thanks to those programmers who did unusual things with their buttons, reported bugs, and helped make Default more robust.

History

1.0 — first release version
1.1 — fixed bug with HideControl and default buttons
1.2 — updated to support new 32-bit clean CDEF messages
1.3 — fixed bug with DragControl
1.4 — fixed bug converting between default buttons and normal buttons;
improved outlines for unusually sized default buttons
1.5 — forced ResEdit to refresh default buttons correctly
2.0 — added automatic Return and Enter support for default buttons;
added window font feature for push buttons, check boxes, and radio buttons;
restructured code to reduce size of CDEF
2.1 — updated to follow Apple's new way of drawing default outlines

Distribution

Lim Unlimited
Postal: 330 W. Iris, Stockton, CA 95210, U.S.A.
Internet: lim@iris.ucdavis.edu
America Online: LimUnltd
CompuServe: 72647,660

```
/*
   Default is a CDEF written using THINK C. This CDEF replaces the standard CDEF 0 and
   simply draws a default button outline and calls the standard CDEF 0 to handle
   everything else.

   © Lim Unlimited, 9 Jul 1990 - All Rights Reserved
*/



/*
   header files
*/

#include <Color.h>
#include <ColorToolbox.h>
#include <FontMgr.h>



/*
   constants and macros
```

```
*/
```

```
#define RESEDIT  1      /* whether to compile ResEdit updating code */

#define NIL      ((void *) 0)

#define INACTIVE 255

#define STANDARD_CDEF_ID   0

enum {                  /* new 32-bit clean messages */
    calcCntlRgn = 10,
    calcThumbRgn
};

#define LO_3_BYTES  0x00FFFFFF

#define SYS_ENVIRONS_VERSION     1

#define DEFAULT_FLAG       '@'
#define USEWFONT_FLAG   'ƒ'

#define OUTLINE_THICKNESS  3
#define OUTLINE_INSET      (OUTLINE_THICKNESS + 1)
#define OUTLINE_OUTSET     (-OUTLINE_INSET)
#define CURVE_ADJUSTMENT   2

#define RECT_IN_RECT(r1, r2)
            \
    ((r1)->top >= (r2)->top && (r1)->left >= (r2)->left &&                \
     (r1)->bottom <= (r2)->bottom && (r1)->right <= (r2)->right)



/*
    typedefs
*/

typedef struct {
    ControlHandle itmHndl;
    Rect             itmRect;
    Byte             itmType;
    Byte             itmData[1];
} Item, *ItemPtr;

typedef struct {
    short     dlgMaxIndex;
    Item      item[1];
} ItemList, *ItemListPtr, **ItemListHdl;

typedef struct {
    Handle    standardCDEF;
    unsigned      has128KROMS:1;
    unsigned      hasColorQD:1;
    unsigned      dialogButton:1;
    unsigned      dialogDefault:1;
#if RESEDIT
    unsigned      inResEdit:1;
#endif
} DefaultData, *DefaultDataPtr, **DefaultDataHdl;



/*
    routines
```

```
*/
```

```
PASCAL long         main                                    (short, ControlHandle, short,
long);



/*
   static routines
*/

static long         CallStandardCDEF           (short, ControlHandle, short, long);
static Boolean      RealHandle                        (long, Boolean);
static short     FindItemNum                          (DialogPeek, ControlHandle);



/*
   static variables
*/

static unsigned char       Copyright[] = "Default 2.1, ©1990 Lim Unlimited";



/*
   main draws the default button outline and calls the standard button CDEF.
*/

PASCAL long main(varCode, theControl, message, param)

register short             varCode;
register ControlHandle  theControl;
register short             message;
register long           param;

{
   register DefaultDataHdl       defaultDataHdl;
   unsigned char                 *title;
   DialogPeek                           theDialog;
   Boolean                              isDefault, useWindowFont, dialogDefault;
   short                                oldRefNum;
   Handle                        standardCDEF;
   SysEnvRec                     sysEnvirons;
   Rect                                 button;
   AuxCtlHndl                           auxCtlHdl;
   RGBColor                             oldColor, frameColor;
   PenState                             penState;
   Pattern                              gray;
   RgnHandle                     outlineRgn;
   register short                       curve;
   register long                 result;
#if RESEDIT
   RgnHandle                     visRgn, oldVisRgn;
   Rect                                 visButton, visBounds;
#endif

   title = (*theControl)->contrlTitle;
   if (!(varCode & ~useWFont) && title[title[0]] == (unsigned char) DEFAULT_FLAG) {
      isDefault = TRUE;
      --title[0];
   } else {
      isDefault = FALSE;
   }
   if (useWindowFont = (title[title[0]] == (unsigned char) USEWFONT_FLAG)) {
```

```
        varCode |= useWFont;
        --title[0];
    }
```

```
       theDialog = (DialogPeek) (*theControl)->
;

   if (message == initCntl) {
       defaultDataHdl = (DefaultDataHdl) NewHandle(sizeof(DefaultData));
       (*theControl)->contrlData = (Handle) defaultDataHdl;

       oldRefNum = CurResFile();
       UseResFile(0);
       standardCDEF = GetResource('CDEF', STANDARD_CDEF_ID);
       (*defaultDataHdl)->standardCDEF = standardCDEF;
       UseResFile(oldRefNum);
       HNoPurge(standardCDEF);

       result = SysEnvirons(SYS_ENVIRONS_VERSION, &sysEnvirons);
       (*defaultDataHdl)->has128KROMS = (sysEnvirons.machineType >= envMachUnknown);
       (*defaultDataHdl)->hasColorQD = sysEnvirons.hasColorQD;
       if (!(varCode & ~useWFont) &&
           RealHandle((long) theDialog->items, !result) &&
           RealHandle((long) theDialog->textH, !result) &&
           theDialog->editField >= -1 &&
           theDialog->editField <= (*(ItemListHdl) theDialog->items)->dlgMaxIndex) {
          (*defaultDataHdl)->dialogButton = TRUE;
       } else {
          (*defaultDataHdl)->dialogButton = FALSE;
       }
       (*defaultDataHdl)->dialogDefault = FALSE;

#if RESEDIT
       /* ResEdit is being used if the control's window has no controls attached */
       if (!theDialog->window.controlList && theDialog == (DialogPeek) FrontWindow()) {
          (*defaultDataHdl)->inResEdit = TRUE;
       } else {
          (*defaultDataHdl)->inResEdit = FALSE;
       }
#endif

       result = CallStandardCDEF(varCode, theControl, message, param);

   } else {
       defaultDataHdl = (DefaultDataHdl) (*theControl)->contrlData;

       /* set default button item number for Dialog Manager */
       if ((*defaultDataHdl)->dialogButton) {
          dialogDefault = (isDefault && !(*theControl)->contrlHilite);
          if (dialogDefault && !(*defaultDataHdl)->dialogDefault) {
             theDialog->aDefItem = FindItemNum(theDialog, theControl);
             if (theDialog->aDefItem) {
                (*defaultDataHdl)->dialogDefault = TRUE;
             }
          } else if (!dialogDefault && (*defaultDataHdl)->dialogDefault) {
             if (theDialog->aDefItem == FindItemNum(theDialog, theControl)) {
                theDialog->aDefItem = 0;
             }
             (*defaultDataHdl)->dialogDefault = FALSE;
          }
       }

       if (message == drawCntl) {

          result = CallStandardCDEF(varCode, theControl, message, param);
          if (isDefault && (*theControl)->contrlVis) {
#if RESEDIT
```

```
/* portions of the ResEdit window are invalidated under certain
   circumstances so it will update correctly */
if ((*defaultDataHdl)->inResEdit) {
```

```
                visRgn = theDialog->window.port.visRgn;
                visBounds = (*visRgn)->rgnBBox;
                button = (*theControl)->contrlRect;
                if (SectRect(&button, &theDialog->window.port.portRect, &visButton) &&
                     RECT_IN_RECT(&visButton, &visBounds)) {
                    InsetRect(&button, OUTLINE_OUTSET, OUTLINE_OUTSET);
                    if (SectRect(&button, &theDialog->window.port.portRect, &visButton) &&
                         !RECT_IN_RECT(&visButton, &visBounds)) {
                        outlineRgn = NewRgn();
                        oldVisRgn = NewRgn();
                        CopyRgn(visRgn, outlineRgn);
                        CopyRgn(visRgn, oldVisRgn);
                        RectRgn(visRgn, &theDialog->window.port.portRect);
                        InsetRgn(outlineRgn, OUTLINE_OUTSET, OUTLINE_OUTSET);
                        DiffRgn(outlineRgn, oldVisRgn, outlineRgn);
                        EraseRgn(outlineRgn);
                        CopyRgn(oldVisRgn, visRgn);
                        DisposeRgn(oldVisRgn);
                        InvalRgn(outlineRgn);
                        DisposeRgn(outlineRgn);
                    }
                }
            }
#endif

            if ((*defaultDataHdl)->hasColorQD) {
                (void) GetAuxCtl(theControl, &auxCtlHdl);
                GetForeColor(&oldColor);
                if (auxCtlHdl) {
                    frameColor = (*(*auxCtlHdl)->acCTable)->ctTable[cFrameColor].rgb;
                    RGBForeColor(&frameColor);
                }
            }

            GetPenState(&penState);
            PenNormal();
            PenSize(OUTLINE_THICKNESS, OUTLINE_THICKNESS);
            if ((*theControl)->contrlHilite == INACTIVE) {
                *(unsigned long *) &gray[0] = *(unsigned long *) &gray[4] = 0xAA55AA55;
                PenPat(&gray);
            }
            button = (*theControl)->contrlRect;
            InsetRect(&button, OUTLINE_OUTSET, OUTLINE_OUTSET);
            curve = ((button.bottom - button.top) >> 1) + CURVE_ADJUSTMENT;
            FrameRoundRect(&button, curve, curve);
            SetPenState(&penState);

            if ((*defaultDataHdl)->hasColorQD) {
                RGBForeColor(&oldColor);
            }
        }

    } else if (message == calcCRgns ||
                message == calcCntlRgn ||
                message == calcThumbRgn) {
        result = CallStandardCDEF(varCode, theControl, message, param);
        if (isDefault && ((message == calcCRgns && param > 0) ||
                            message == calcCntlRgn)) {
            OpenRgn();
            button = (*theControl)->contrlRect;
            InsetRect(&button, OUTLINE_OUTSET, OUTLINE_OUTSET);
            curve = ((button.bottom - button.top) >> 1) + CURVE_ADJUSTMENT;
            FrameRoundRect(&button, curve, curve);
```

```
        CloseRgn((RgnHandle) param);
    }
```

```
        } else if (message == dispCntl) {
            result = CallStandardCDEF(varCode, theControl, message, param);

            /* MultiFinder loads and shares system resources in the system heap; make
                standard button CDEF purgeable only if it is in the application heap */
            standardCDEF = (*defaultDataHdl)->standardCDEF;
            if (HandleZone(standardCDEF) == ApplicZone()) {
                HPurge(standardCDEF);
            }
            DisposHandle(defaultDataHdl);

#if 0
        /* messages not used by Default but documented for completeness */
        } else if (message == testCntl ||
                    message == posCntl ||
                    message == thumbCntl ||
                    message == dragCntl ||
                    message == autoTrack) {
            result = CallStandardCDEF(varCode, theControl, message, param);
#endif

        /* pass along messages which are not used by Default or are currently undefined */
        } else {
            result = CallStandardCDEF(varCode, theControl, message, param);
        }
    }

    if (useWindowFont) {
        ++(*theControl)->contrlTitle[0];
    }
    if (isDefault) {
        ++(*theControl)->contrlTitle[0];
    }
    return(result);
}



/*
    CallStandardCDEF calls the standard button CDEF.
*/

static long CallStandardCDEF(varCode, theControl, message, param)

short           varCode;
ControlHandle theControl;
short           message;
long            param;

{
    register DefaultDataHdl       defaultDataHdl;
    register Handle               standardCDEF;
    register SignedByte           flags;
    register long                 result;

    defaultDataHdl = (DefaultDataHdl) (*theControl)->contrlData;

    /* load standard button CDEF if it was purged */
    standardCDEF = (*defaultDataHdl)->standardCDEF;
    if (!*standardCDEF) {
        LoadResource(standardCDEF);
        HNoPurge(standardCDEF);
    }
```

```
/* save and restore handle state just in case control is reentrant */
if ((*defaultDataHdl)->has128KROMS) {
```

```
        flags = HGetState(standardCDEF);
    } else {
        flags = (long) *standardCDEF >> 24;
    }
    HLock(standardCDEF);
    result = CallPascalL(varCode, theControl, message, param, *standardCDEF);
    if ((*defaultDataHdl)->has128KROMS) {
        HSetState(standardCDEF, flags);
    } else {
        *standardCDEF = (Ptr) (((long) *standardCDEF & LO_3_BYTES) | (flags << 24));
    }
    return(result);
}

/*
    RealHandle returns whether the given address is a handle in the system heap or
    application heap.
*/

static Boolean RealHandle(addr, hasStripAddr)

register long addr;
Boolean          hasStripAddr;

{
    register Boolean real;
    register THz     sysZone, applZone, heapZone;
    real = FALSE;
    addr = (hasStripAddr) ? StripAddress(addr) : addr & LO_3_BYTES;
    if (addr && !(addr & 1)) {
        sysZone = SystemZone();
        applZone = ApplicZone();
        if (((addr >= (long) &sysZone->heapData && addr < (long) sysZone->bkLim) ||
              (addr >= (long) &applZone->heapData && addr < (long) applZone->bkLim)) &&
             *(long *) addr && !(*(long *) addr & 1)) {
            heapZone = HandleZone(addr);
            if (!MemError() && (heapZone == sysZone || heapZone == applZone)) {
                real = TRUE;
            }
        }
    }
    return(real);
}




/*
    FindItemNum returns the item number of the given control in the given dialog.
*/

static short FindItemNum(theDialog, theControl)

DialogPeek       theDialog;
ControlHandle theControl;

{
    register short      numItems, itemNum;
    register Ptr     itemPtr;

    numItems = (*(ItemListHdl) theDialog->items)->dlgMaxIndex + 1;
    itemPtr = (Ptr) (*(ItemListHdl) theDialog->items)->item;
```

```
for (itemNum = OK; itemNum <= numItems; ++itemNum) {
   if (((ItemPtr) itemPtr)->itmHndl == theControl) return(itemNum);
   itemPtr += sizeof(Item) + (((long) ((ItemPtr) itemPtr)->itmData[0] + 1) & ~1);
```

```
    }
    return(0);
}
```

ToolBox Gotchas

by John Norstad

While developing Disinfectant I ran into a number of "gotchas" that caused me great grief. I thought it would be nice to tell the rest of you about these problems, in the hope that you'll be able to avoid them in your own programs. I've told DTS about most of this stuff.

Gotcha #1. Watch out for PBGetCatInfo calls with TOPS.

The file manager routine PBGetCatInfo uses a parameter block of type CInfoPBRec. Make certain that you pass a pointer to the full parameter block when using MPW C, even if you know in advance that the object is a directory. Don't just allocate and pass a pointer to the DirInfo variant. The DirInfo variant is four bytes shorter than the full union type, and with TOPS the PBGetCatInfo call sets those four bytes at the end. If your parameter block is not big enough you'll trash the stack.

Gotcha #2. Make certain you're in the proper heap zone before calling ReleaseResource.

At the bottom of IM II-26, in the Memory Mangler chapter, is the warning "Be sure, when calling routines that access blocks, that the zone in which the block is located is the current zone." Heed this warning, especially when releasing resources. Bob Hablutzel and I discovered (after hours in Macsbug) that on the 128K ROMs, if you try to release an empty (unloaded) resource in the system heap, and if you neglect to set the current zone to the system zone, then the system will trash the free master pointer list. This is not good, and will almost undoubtedly lead to subsequent bizarre behavior.

Here's the code I use to release a resource:

```
curZone = GetZone();
SetZone(HandleZone(theRez));
ReleaseResource(theRez);
SetZone(curZone);
```

Gotcha #3. Don't believe Inside Macintosh. (HandleZone)

On page IM II-34 we read the following warning in the description of the HandleZone routine: "If handle h is empty (points to a NIL master pointer), HandleZone returns a pointer to the current heap zone." This is false - HandleZone properly returns a pointer to the heap zone that contains the master pointer. See Gotcha #2 above.

Gotcha #4. Don't expect OpenResFile to do sanity checking.

Neither OpenResFile nor OpenRFPerm does any sanity checking of any sort when opening a resource file. If the file is damaged or contains trash it is very possible for the Resource Mangler to bomb or hang inside the OpenResFile or OpenRFPerm call. Often what happens is that it makes a Memory Mangler request for some ridiculously huge block of memory. If you have a GrowZone proc this can cause problems.

To prevent this problem you must write a sanity checker of your own that opens the resource

file as a binary file and checks at least the most important structural characteristics of the file. If your sanity check fails you must avoid calling OpenResFile or OpenRFPerm on the file. In Disinfectant I check that the resource map and resource data are within the logical EOF of the file and don't overlap, I check that the resource type list immediately follows the resource map, and I check that the resource name list starts within the logical eof.

DTS tells me that the only way to be completely safe is to do a complete sanity check of the entire resource fork - e.g., rewrite the RezDet MPW tool.

Damaged and trashed resource forks are much more common than you might think.

Gotcha #5. Don't believe Inside Macintosh. (OpenResFile)

In the description of the OpenResFile routine, IM I-115 states "If the resource file is already open, it doesn't make it the current resource file; it simply returns the reference number." This is false. If the resource file is already open, OpenResFile in fact DOES make it the current resource file. OpenRFPerm also has the same behavior, in those cases where OpenRFPerm returns the reference number of the previously opened copy of the file, rather than opening a new access path (see IM IV-17 and TN 185).

Gotcha #6. Watch out for Standard File if you unmount volumes.

The standard file package keeps track of the last volume it used in the low core global SFSaveDisk, which contains the negative of the vol ref num of the last volume used. If your program unmounts this volume and then later calls the standard file package again, it will post an alert saying that "A system error has occurred. Please try again." A simple fix for this problem is to check the vRefNum stored in SFSaveDisk immediately before any calls to standard file. Call PBGetVInfo to see if the volume still exists. If it doesn't, make an indexed call to PBGetVInfo to get the vRefNum of the first volume in the VCB queue, and set SFSaveDisk to the negative of this vRefNum. Also set CurDirStore to fsRtDirID.

Gotcha #7. Don't believe Inside Macintosh.

IM I-116 states that "When calling the CurResFile and HomeResFile routines, described below, be aware that for the system resource file the actual reference number is returned." This is false. CurResFile does indeed return the actual reference number of the system file (2), but HomeResFile in fact returns 0 for system file resources.

Gotcha #8. Don't believe Inside Macintosh.

IM I-126 states "Like the attributes of individual resources, resource file attributes are specified by bits in the low-order byte of a word." This is false. In fact, the resource file attributes are stored in the high-order byte of the word.

Gotcha #9. Directory IDs are longs, not shorts, stupid.

Directory IDs, unlike volume reference numbers and working directory ids, are longs, not shorts. Watch out for this one. It's really easy to declare a dirID to be a short by mistake, and unless you're using Modula-2 you probably won't catch the bug even with extensive beta testing. Don't feel too stupid if you do this - I have it on good authority that ResEdit once had this bug!

Gotcha #10. Always set the ioNamePtr field in file manager param blocks.

See TN 179. Read it. Believe it. Always set ioNamePtr. Set it to nil if you don't care about

the name. I made this mistake three times while developing Disinfectant, and all three times it took FOREVER to find the bug. The problem is those silly little arrows in the file manager chapter of IM IV. They all point to the left for ioNamePtr, which usually means that you don't have to set the field before calling the routine.

I hope my experiences help somebody.

John Norstad Academic Computing and Network Services Northwestern University

How do you Hide the menu bar?

by: Earle R. Horton

The code I posted yesterday would cause problems in the unlikelyevent that your program would crash while the Menu Bar was hidden. Specifically, it would replace GrayRgn with a handle to a Region in the application heap, and save the real GrayRgn Handle for restoration later when you restored the Menu Bar. If your program crashed, and the Menu Bar was hidden, then GrayRgn was left pointing to a Region in a defunct application heap, which could cause all sorts of problems for applications which were still running.

This version does not change the Handle, but rather modifies the contents of GrayRgn. If you crash with the Menu Bar hidden, GrayRgn is left pointing to a valid area of storage, at least. There are still problems since the Menu Bar can be left hidden, but these are slightly less severe than leaving a dangling Handle in the system heap.

Earle

```
{
This unit provides the ability to hide the Menu Bar, and to show
it.  When the Menu Bar is hidden, windows may be placed in the
area normally used for the Menu Bar.

Usage:
       PROCEDURE MBar_Init:
         Used to initialize global variables used here.
         Call once at beginning of application code.
       PROCEDURE MBar_be_Gone;
         Hides the Menu Bar.
       PROCEDURE MBar_Restore;
         Shows the Menu Bar.  Call this whenever you are going
         to be put into the background.  Call before Exit.

The procedures in this unit will probably break under some future
release of the Macintosh Operating System, because they manipulate
the GrayRgn.  They do work under MultiFinder 6.1a2.  The most
serious warning I can give concerning use of these routines is
that you must never allow your application to be placed into the
background with the Menu Bar hidden.

Possible compatibility problems:

 Modifies lowmem MBarHeight
 Modifies contents of GrayRgn

This file is part of Earle R. Horton's private source code library.
Earle R. Horton assumes no responsibility for any damages arising
out of use of this source code for any purpose.  Earle R. Horton
places no restrictions on use of all or any part of this source code,
except that this paragraph may not be altered or removed.

Original Language:
  MPW Pascal, v. 2.0.2
Origination Date:
  March 20, 1989
Modifications:
  April 4, 1989 ERH
    First version changed lowmem GrayRgn to point to a Region in the
application heap while the Menu Bar was hidden.  This version copies the
new Region to GrayRgn.  If the application crashes with the Menu Bar
hidden, GrayRgn no longer points to part of the defunct application
heap.  There are still problems, however, because the Menu Bar is left
```

```
hidden and other applications cannot access it.
}
UNIT MenuBar;
 INTERFACE
```

```
   USES
     {$Load PasDump.dump}
     Memtypes, Quickdraw, OSIntf, Script, ToolIntf;

   PROCEDURE MBar_be_Gone;
   PROCEDURE MBar_Restore;
   PROCEDURE MBar_Init;

   PROCEDURE SetMBarHeight(newheight:integer);
   INLINE smPopStack2Word,smMBarHeight;        { move.w        (a7)+,$0BAA }

   VAR
     Real_MBar_Height: integer;        { Copy of lowmem MBarHeight }
     Save_Region:      RgnHandle;      { Copy of GrayRgn }
     Hidden_Flag:      Boolean;        { State info }
     MBar_Rect:        Rect;           { Rect in which MBar is drawn }

 IMPLEMENTATION

   PROCEDURE MBar_Init;
   BEGIN
     Hidden_FLag := false;
     Real_MBar_Height := GetMBarHeight;
     SetRect(MBar_Rect,
             screenBits.bounds.left,
             screenBits.bounds.top,
             screenBits.bounds.right,
             screenBits.bounds.top + Real_MBar_Height);
   END;
{
Make the Menu Bar go away under MultiFinder or UniFinder.
Since this procedure manipulates the Gray Region, future
compatibility is unknown.
}
   PROCEDURE MBar_be_Gone;
   VAR
     MBar_Region:      RgnHandle;
     theWindow:        WindowPeek;
     BEGIN
       IF not Hidden_Flag THEN
         BEGIN
           Hidden_Flag := true;
                                              { Get some Regions to work with}
           Save_Region := NewRgn;
           MBar_Region := NewRgn;
                                            { Set the Menu Bar height to zero}
           SetMBarHeight(0);
}
           CopyRgn(GetGrayRgn,Save_Region);
              { Fix up GrayRgn to cover the old GrayRgn plus the Menu Bar Rect}
           RectRgn(MBar_Region,MBar_Rect);
           UnionRgn(GetGrayRgn,MBar_Region,GetGrayRgn);
                   { Paint and fix up visRgn for any windows with exposed area}
           theWindow := WindowPeek(FrontWindow);
           PaintOne(theWindow,MBar_Region);
           PaintBehind(theWindow,MBar_Region);
           CalcVis(theWindow);
           CalcVisBehind(theWindow,MBar_Region);
           DisposeRgn(MBar_Region);                          { Clean up, leave}
         END;
     END;
{
```

```
Restore the Menu Bar and GrayRgn to normality.
Call when app is put into background.
}
```

```
   PROCEDURE MBar_Restore;
   VAR
     theWindow:        WindowPeek;
   BEGIN
   IF Hidden_Flag THEN
     BEGIN
       Hidden_Flag := false;
                          { Restore to original }
       CopyRgn(Save_Region,GetGrayRgn);
                       { Restore Menu Bar height }
       SetMBarHeight(Real_MBar_Height);
                    { Fix up any covered windows }
       RectRgn(Save_Region,MBar_Rect);
       theWindow := WindowPeek(FrontWindow);
       CalcVis(theWindow);
       CalcVisBehind(theWindow,Save_Region);
       DisposeRgn(Save_Region);
               { Draw the Menu, get out of here }
       HiliteMenu(0);
       DrawMenuBar;
     END;
   END;
END.
```

BitMap Rotation in C (and support routines)

by Juri Munkki

I'm including two small Think C programs along with sources, project files and resources.

I tried to make it as easy as possible to include these programs with the usenet programmer's guide. The resource files are extremely simple. You only need one window resource and one picture resource. The windows should have id 1000 and they should be visible, but all the other parameters can be whatever you want. The PICTs should have id 1000 and can be just about anything.

The other program documents the internal region data format by providing the functionality of the BitmapToRegion call. This may actually be useful to those programmers who wish to have their programs running on machines without 32 bit QD, although I recommend using Apple's licenseable code instead of mine. In addition, understanding the region data format will allow programmers a better understanding of quickdraw algorithms.

The other program performs a function that is not available from Apple or from anywhere else that I know of. So far programs have had to have their own bitmap rotation routines. A 90 degree rotation routine was already included with the guide, but this routine allows free rotation of any bitmap. Current performance is limited by the drawing speed. To optimize this routine, draw into an offscreen bitmap using your own commands. If there are enough requests for this, I could write something to do it more efficiently. Using PaintRect or MoveTo/LineTo is definitely not the way to do it.

I hope you find these interesting.

[Juri's code lined up beutifully until it was imported into Word.]

{---------- Bit map Rotation ----------}

```
/*
>>   BitRot.c   Bitmap Rotation Algorithm Tester.
>>            Copyright (c)1990, Juri Munkki
>>            Permission to use is granted for noncommercial applications.
>>
>>   This program rotates a bitmap to any angle. With modifications, it
>>   can be used to scale as well as rotate.
>>
>>   The idea is to perform the inverse tranformation to the destination
>>   bitmap. The trick is to avoid multiplication in the main loop. This
>>   program also optimizes so that it doesn't copy any extra pixels.
>>
>>   The routine does a simplified flood fill to copy every pixel in the
>>   destination from the source. There are no guarantees that every pixel
>>   from the source is used, but every pixel in the destination is checked.
>>
>>   The program works it's way from near the center of the rotated rectangle.
>>   To illustrate:
>>           .            .            .            .
>>          / \          / \          /_\          /_\
>>         /   \        /   \        /___\        /___\
>>        /  .  \      /_____\      /_____\      /_____\
>>        \     /      \     /      \     /      \_____/
>>         \   /        \   /        \   /        \___/
>>          \ /          \ /          \ /          \_/
>>           '            '            '            '
>>
>>   The fill operation starts from the center of the rectangle and works it's
>>   way up and down. While on the way, the point may also move left or right
>>   depending on a displacement value that is calculated beforehand. The fill
>>   always ends at topmost and bottommost corners. To Try out how the fill works,
>>   use a relatively dark picture and/or add some delays in the plotting subroutine.
>>
>>   To compile:
```

```
>>      long is 32 bits
>>      fixed point numbers are 16+16 bit.
```

```
>>      integers are 16 bits.
>>      Needs a 'WIND' id 1000 window template resource.
>>      Needs a 'PICT' id 1000 as the picture to rotate.
>>
>>   Limitations: For extremely large bitmaps, there might be a problem with fixed
>>   point resolution. To improve resolution, use larger fixed point numbers or
>>   extended precision floating point.
*/

#define   PIC_ID   1000         /* Picture resource ID 1000 is used as the demo picture  */

/*   Prototypes:
*/
void   PictBit(BitMap *,int);    /*  Creates a bitmap and draws the picture into
it.         */
int      origpixel(long,long);   /*  Almost same as GetPixel, but optimized for our
purposes   */

typedef   struct        /*   A Handy structure that keeps track of coordinates      */
{                       /*   in the source and destination rectangles.            */
  int      x,y;        /*   Location on destination bitmap in integral coordinates   */
  long   xo,yo;        /*   Location on source bitmap in fractional coordinates      */
}   place;

long   maxx,maxy;       /*   Size of source bitmap as a fractional (16:16) number.   */

WindowPtr   mywind;     /*   Any port in a storm for drawing the rotated bitmap.      */
BitMap      mybits;     /*   Bitmap to hold the source bits.                        */
long      sinr,cosr;    /*   Sin and Cosine values.       65536==1.0               */

/*
>>   ScanLeftRight tries to go as far left in the destination as the source
>>   rectangle permits. While doing this, it copies the pixels from the source
>>   to the destination. The same thing is done from the center to the right.
*/
void   ScanLeftRight(start)
place   *start;
{
  place   left,right;

  left=right=*start;               /*   Starting point.         */
  if(origpixel(left.xo,left.yo))   /*   Copy starting point.   */
  {   PlotDot(left.x,left.y);
  }

  /*   Check source rectangle boundaries.                  */
  while(left.xo>=0 && left.yo>=0 && left.xo<maxx && left.yo<maxy)
  {   if(origpixel(left.xo,left.yo))
     {   PlotDot(left.x,left.y);   /*   Copy pixel.            */
     }
     left.x--;                 /*   Move left.            */
     left.xo-=cosr;            /*   Move within source      */
     left.yo-=sinr;
  }

  right.x++;                  /*   Move right.            */
  right.xo+=cosr;
  right.yo+=sinr;

  /*   Check source rectangle boundaries.                  */
  while(right.xo>=0 && right.yo>=0 && right.xo<maxx && right.yo<maxy)
  {   if(origpixel(right.xo,right.yo))
     {   PlotDot(right.x,right.y);
```

```
    }
    right.x++;                /*   Move right            */
    right.xo+=cosr;
```

```
      right.yo+=sinr;
   }
}
void   main()
{
   long      theangle;             /*   Angle of rotation.              */
   place      goup,godown;         /*   Coordinates.                    */
   int         disp,edgex;         /*   Displacement (see below).       */
   Point      mouse,oldmouse;      /*   mouse locations for test.       */

   InitGraf(&thePort);      InitCursor();
   InitFonts();         InitWindows();
   InitMenus();         TEInit();
   InitDialogs(0L);                /*   Start up managers.             */
   mywind=GetNewWindow(1000,0,-1); /*   Open up a window.              */
   SetPort(mywind);                /*   Draw in this new window.       */

   PictBit(&mybits,PIC_ID);      /*   Read the picture into a bitmap.         */
   SetupGetPixel();              /*   Prepare for fast read of bitmap.      */

   maxx=((long)mybits.bounds.right)<<16;   /*   Source boundaries are changed   */
   maxy=((long)mybits.bounds.bottom)<<16;   /*   into fixed point numbers      */

   while(!Button())            /*   Quit when button is down.            */
   {   GetMouse(&mouse);         /*   Find out mouse location.            */
      if(mouse.h!=oldmouse.h)      /*   Has horizontal position changed?      */
      {   oldmouse.h=mouse.h;
         theangle=mouse.h*1024L;   /*   Rotation angle from horizontal value.   */

         EraseRect(&mywind->portRect);   /*   Erase window contents.         */

         sinr=FracSin(-theangle)>>14;   /*   Sin for inverse rotation.      */
         cosr=FracCos(-theangle)>>14;   /*   Cosine for inverse rotation.   */

         goup.x=mywind->portRect.right/2; /*   Destination center x coordinate.*/
         goup.y=mywind->portRect.bottom/2;   /*   --   ''   --      y coordinate.*/
         goup.xo=(long)mybits.bounds.right<<15;   /*   Center of source bitmap.*/
         goup.yo=(long)mybits.bounds.bottom<<15;   /*   Center of source bitmap.*/

         godown=goup;            /*   copy center to "godown".         */

         /*   Adjust starting position according to rectangle size and angle.
         **   Basically we transform one corner of the rectangle to find out
         **   where the fill should end.
         */
         if(cosr*sinr>0) disp=(-mybits.bounds.right*cosr + mybits.bounds.bottom*sinr)>>17;
         else  disp=(-mybits.bounds.right * cosr - mybits.bounds.bottom * sinr) >> 17;

         if(sinr>0)      disp=-disp;

         ScanLeftRight(&goup);   /*   Copy first line to destination.         */

         edgex= (disp>0) ? disp : -disp;   /*   edgex=ABS(disp)               */

         /*   Go up until source rectangle bound is crossed.               */
         do
         {   while(goup.xo>=0 && goup.yo>=0 && goup.xo<maxx && goup.yo<maxy)
            {   ScanLeftRight(&goup);

               goup.y--;      /*   Go up.                              */
               goup.xo+=sinr;   /*   Move in source bitmap coordinates.      */
               goup.yo-=cosr;
            }
```

```
if(disp>0)          /*   Stay inside bounds as long as possible.   */
{   goup.x++;       /*   This is done by adjusting the location   */
```

```
               goup.xo+=cosr;   /*   of the fill.                    */
               goup.yo+=sinr;
            }
            else
            {   goup.x--;
               goup.xo-=cosr;
               goup.yo-=sinr;
            }
         }   while(edgex-- > 0);   /*   Adjust only as long as it is useful.   */

         edgex= (disp>0) ? disp : -disp;   /*   edgex=ABS(disp)              */

         godown.y++;            /*   Go down.                        */
         godown.xo-=sinr;
         godown.yo+=cosr;

         /*   Go down until source rectanlge bound is crossed.           */
         do
         {   while(godown.xo>=0 && godown.yo>=0 && godown.xo<maxx && godown.yo<maxy)
            {   ScanLeftRight(&godown);

               godown.y++;
               godown.xo-=sinr;
               godown.yo+=cosr;
            }

            if(disp<0)
            {   godown.x++;
               godown.xo+=cosr;
               godown.yo+=sinr;
            }
            else
            {   godown.x--;
               godown.xo-=cosr;
               godown.yo-=sinr;
            }
         }   while(edgex-- > 0);
      }
   }
}
```

{------------------ Bit Support for rotation ----------------------------------}

```
/*
>>  BitSupport.c  Bitmap Rotation Algorithm Tester.
>>          Support routines for bitmap rotation
>>          Copyright ©1990, Juri Munkki
>>          Permission to use is granted for noncommercial applications.
>>
>>  Ugly routines to test and set pixel values. You should start by optimizing
>>  these routines, if you wish to increase the speed of this program. The
>>  PlotDot routine is the real bottleneck of this program. Origpixel is quite
>>  fast, since it doesn't use toolbox routines.
*/

extern  BitMap    mybits;
   Ptr      *index;

void  PlotDot(x,y)
int    x,y;
{
 Rect  foo;
```

```
foo.left=x;
foo.right=x+1;
foo.top=y;
```

```
 foo.bottom=y+1;
 PaintRect(&foo);
}

int    origpixel(x,y)
long  x,y;
{
asm  {  move.w  y,D0
   asl.w  #2,D0
   move.l  index,A0
   move.l  0(A0,D0),A0
   move.w  x,D0
   move.w  D0,D1
   lsr.w  #3,D0
   add.w  D0,A0
   moveq.l  #7,D0
   and.w  D0,D1
   sub.w  D1,D0
   btst  D0,(A0)
   beq    @nothing
   moveq.l  #-1,D0
   return
@nothing
   clr.w  D0
   return
 }
/*  return BitTst(index[y>>16],x>>16);*/
}

void  SetupGetPixel()
{
 int    i;
 Ptr    base;

 index=(Ptr *)NewPtr(mybits.bounds.bottom*sizeof(long));
 base=mybits.baseAddr;
 for(i=0;i<mybits.bounds.bottom;i++)
 {  index[i]=base;
   base+=mybits.rowBytes;
 }
}

/*  PictBit reads a picture resource, creates
>>  a large enough bitmap and draws the picture
>>  into it. The Bits bitmap is supplied to the
>>  routine. Space for the actual bits is reserved
>>  with NewPtr. Be sure to deallocate it once
>>  it is no longer needed!
>>
>>  No error checking is made.
*/
void  PictBit(Bits,PictId)
BitMap  *Bits;
int    PictId;
{
 GrafPort  AnyPort;
 GrafPtr    SavedPort;
 PicHandle  ThePic;
 Rect    TempRect;
 long    RAMNeeded;

 GetPort(&SavedPort);
 OpenPort(&AnyPort);
```

```
ThePic=(PicHandle)GetResource('PICT',PictId);
TempRect=(*ThePic)->picFrame;
```

```
 OffsetRect(&TempRect,-TempRect.left,-TempRect.top);
 Bits->bounds=TempRect;

 Bits->rowBytes=((TempRect.right + 15) >> 4) << 1;  /*  Round to word boundary  */
 RAMNeeded=Bits->rowBytes*TempRect.bottom;      /*  Calculate RAM for bits  */
 Bits->baseAddr=NewPtr(RAMNeeded);

 SetPortBits(Bits);
 AnyPort.portRect=TempRect;
 RectRgn(AnyPort.visRgn,&TempRect);
 RectRgn(AnyPort.clipRgn,&TempRect);

 EraseRect(&TempRect);
 DrawPicture(ThePic,&TempRect);

 ReleaseResource(ThePic);
 ClosePort(&AnyPort);
 SetPort(SavedPort);
}
```

{---------- Bit map to region ----------}

```
/*
>>  BitRegion.c, 04/23/89                          <<
>>  My routine for converting a bitmap into a region.          <<
>>                                  <<
>>  Juri Munkki, jmunkki@kampi.hut.fi                   <<
>>  Senior Systems Analyst                          <<
>>  Helsinki University of Technology Computing Centre          <<
>>  Otakaari 1 U044A, SF02150 Espoo, Finland              <<
>>                                  <<
>>  This program is in the PUBLIC DOMAIN, but:              <<
>>    I would really like to join the NeXT registered developer    <<
>>    program. I returned the forms, but I haven't heard anything    <<
>>    from NeXT. Please help me, if you can affect their decision.  <<
>>                                  <<
>>  Known bug:  This program knows how to create regions larger than  <<
>>        32 KB. QD doesn't support anything larger than 32KB.  <<
*/

#define  PIC_ID  1000      /*  Resource ID of test picture    */
RgnHandle  BitRgn(BitMap *);  /*  Function prototype for BitRgn  */

/*  PictBit reads a picture resource, creates
>>  a large enough bitmap and draws the picture
>>  into it. The Bits bitmap is supplied to the
>>  routine. Space for the actual bits is reserved
>>  with NewPtr. Be sure to deallocate it once
>>  it is no longer needed!
>>
>>  No error checking is made.
*/
void  PictBit(Bits,PictId)
BitMap  *Bits;
int    PictId;
{
 GrafPort  AnyPort;
 GrafPtr    SavedPort;
 PicHandle  ThePic;
 Rect    TempRect;
 long    RAMNeeded;
```

```
GetPort(&SavedPort);
OpenPort(&AnyPort);
```

```
 ThePic=(PicHandle)GetResource('PICT',PictId);
 TempRect=(*ThePic)->picFrame;

 OffsetRect(&TempRect,-TempRect.left,-TempRect.top);
 Bits->bounds=TempRect;

 Bits->rowBytes=((TempRect.right + 15) >> 4) << 1;  /*  Round to word boundary  */
 RAMNeeded=Bits->rowBytes*TempRect.bottom;      /*  Calculate RAM for bits  */
 Bits->baseAddr=NewPtr(RAMNeeded);

 SetPortBits(Bits);
 AnyPort.portRect=TempRect;
 RectRgn(AnyPort.visRgn,&TempRect);
 RectRgn(AnyPort.clipRgn,&TempRect);

 EraseRect(&TempRect);
 DrawPicture(ThePic,&TempRect);

 ReleaseResource(ThePic);
 ClosePort(&AnyPort);
 SetPort(SavedPort);
}

/*
>>  Convert a bitmap into a region.
>>  The bitmap origin should be at the top left corner and it
>>  shouldn't be wider than 8192 pixels. You might want to add
>>  some error checks for weird or illegal bitmaps.
*/
RgnHandle  BitRgn(Bits)
BitMap    *Bits;
{
 register Handle  Target;        /*  This is where we write the region  */
 register short  *TargetP;      /*  Pointer to region data array    */
 register long  MaxTarget;      /*  Memory management stuff        */
 register long  CurTarget;      /*  Index into the region data array  */
 long      RowStart;      /*  Index of first x value on row    */
 long      TargetSize,RgnSize;  /*  Size in data words & bytes      */
 Rect      TempRect,RgnBounds;  /*  Temporary & region bounds rects    */
 BitMap      RowBitMap;      /*  Working bitmap with one row in it  */
 char      RowBitData[1024];  /*  Buffer for pixels above (8192 pix)  */
 int        i;          /*  Row counter in a "for" loop      */
 register int  x;          /*  Column counter in a "for" loop    */
 register int  pixelstatus;    /*  Flag is false if last pixel is white*/

 TargetSize=4096;          /*  Initial guess for final region size  */
 Target=NewHandle(TargetSize);    /*  Allocate initial data buffer    */
 if(Target==0)  return 0;      /*  Did we run out of RAM? 0=failure.  */
 TargetP=(short *)(*Target + 10);  /*  TargetP points to region data    */
 HLock(Target);            /*  We just derefenced target. Lock it.  */
 MaxTarget=(TargetSize-20)/2;    /*  A safe maximum value for our index  */
 CurTarget=0;            /*  Start with target index 0 (no data)  */

 /*  Set region bounds to nothing:                    */
 SetRect(&RgnBounds,32767,32767,-32767,-32767);

 /*  Set up a bitmap with a single line:                  */
 TempRect=Bits->bounds;      /*  Set up left & right bounds      */
 TempRect.top=0;          /*  Single row bitmap with top=0    */
 TempRect.bottom=1;        /*  Single row bitmap with bottom=1    */
 RowBitMap.bounds=TempRect;
 RowBitMap.baseAddr=RowBitData;
 RowBitMap.rowBytes=((TempRect.right + 15) >> 4) << 1;
```

```
/*  Start out with the first line of the source bitmap          */
CopyBits(Bits,&RowBitMap,&TempRect,&RowBitMap.bounds,srcCopy,0);
```

```
 for(i=Bits->bounds.bottom;i>=0;i--)      /*  For every line and more      */
 {  TargetP[CurTarget++]=TempRect.top;    /*  Row data starts with Y value  */
   RowStart=CurTarget;            /*  X values on row start here    */
   pixelstatus=0;              /*  Pixels outside bitmap are white  */
   for(x=Bits->bounds.left;x<Bits->bounds.right;x++)
   {  if((BitTst(RowBitData,x)!=0) != pixelstatus)  /*  Test for a change  */
     {  pixelstatus= !pixelstatus;          /*  Color changed    */
       TargetP[CurTarget++]=x;           /*  Record x coordinate  */
       if(CurTarget>=MaxTarget)          /*  Is the buffer full?  */
       {  TargetSize+=2048;            /*  Enlarge the buffer    */
         HUnlock(Target);            /*  Unlock to change size  */
         SetHandleSize(Target,TargetSize);    /*  Change the size      */
         if(MemErr)                /*  No success?        */
         {  DisposHandle(Target);        /*  Dispose of what we have  */
           return 0;              /*  return failure.      */
         }
         TargetP=(short *)(*Target + 10);    /*  Dereference handle    */
         HLock(Target);              /*  Lock it again      */
         MaxTarget=(TargetSize-20)/2;      /*  New maximum index    */
       }
     }
   }
   if(pixelstatus)    TargetP[CurTarget++]=x;  /*  Last pixel was black, record edge  */
   if(RowStart==CurTarget)            /*  Row was empty (no changes)      */
     CurTarget--;              /*  Remove Y value from data      */
   else
   {  /*  Check for new region bounds:                        */
     if(TargetP[RowStart]   <RgnBounds.left)    RgnBounds.left=TargetP[RowStart];
     if(TargetP[CurTarget-1]>RgnBounds.right)  RgnBounds.right=TargetP[CurTarget-1];

     RgnBounds.bottom=TempRect.top;

     TargetP[CurTarget++]=32767;        /*  Write an "end of line" flag      */
   }

   /*  Copy current line into the single line bitmap:              */
   if(i>0) CopyBits(Bits,&RowBitMap,&TempRect,&RowBitMap.bounds,srcCopy,0);

   TempRect.top++; TempRect.bottom++;      /*  Move one line down      */

   /*  If we are still inside the bitmap, XOR this line with the previous line:*/
   if(i>1)  CopyBits(Bits,&RowBitMap,&TempRect,&RowBitMap.bounds,srcXor,0);
 }

 RgnBounds.top=TargetP[0];  /*  Top boundary is first recorded Y coordinate    */

 /*  If the region is empty, set the bounds rect to an empty rectangle:      */
 if(RgnBounds.right<=RgnBounds.left || RgnBounds.bottom<=RgnBounds.top)
   SetRect(&RgnBounds,0,0,0,0);

 TargetP[CurTarget++]=32767;  /*  Write an "end of region" flag          */
 HUnlock(Target);      /*  Unlock our target region.            */
 RgnSize=CurTarget*2+10;    /*  Calculate region size.              */
 if(RgnSize<=28)  RgnSize=10;  /*  Rectangular or empty region is only a Rect    */

 (*(RgnHandle)Target)->rgnBBox=RgnBounds;/*  Store region bounds rectangle    */
 (*(RgnHandle)Target)->rgnSize=RgnSize;  /*  Store region size (low 16 bits)    */
 SetHandleSize(Target,RgnSize);      /*  Resize region to optimally small  */
 return (RgnHandle)Target;        /*  Return resulting region handle    */
}

/*  This is just a short test program "main":
```

```
*/
void  main()
{
```

```
 WindowPtr  TestWindow;    /*  Simple window used for testing  */
 RgnHandle  TheRegion;    /*  Region handle for test region  */
 BitMap    TheBits;    /*  Bitmap for testing        */
 EventRecord  MyEvent;

 /*  "Magic Incantations" (Copyfight Apple Computer, Inc.)  */
 InitGraf(&thePort);    InitCursor();  InitFonts();      InitWindows();
 InitMenus();      TEInit();    InitDialogs(0L);    InitCursor();

 TestWindow=GetNewWindow(1000,0,-1);
 SetPort(TestWindow);

 PictBit(&TheBits,PIC_ID);  /*  Read the picture into a bitmap      */
 TheRegion=BitRgn(&TheBits);  /*  Convert the bitmap into a region    */
 if(TheRegion)        /*  If we get a region, let's play with it  */
 {  InvertRgn(TheRegion);  /*  Display region              */
   FlushEvents(everyEvent,0);
   while(GetNextEvent(mDownMask,&MyEvent)==0);

   GlobalToLocal(&MyEvent.where);
   InvertRgn(TheRegion);  /*  Hide region, then drag it around.    */
   DragGrayRgn(TheRegion,MyEvent.where,
         &TestWindow->portRect,
         &TestWindow->portRect,
         noConstraint,0L);
 }
}
```

INIT Skeleton Code

by Jon Wätte

SetWindow INIT, which lets you place windows anywhere on the screen when an application calls ShowWIndow on it. (Just like TWM under X)

The INIT consists of a loader (that should be compiled as an INIT resource) and two patches (of which the loader will choose to install one depending on color QD availability)

The patches should be compiled as "tpat" code resources, and the b/w version should have resource id 128, the color version id 129.

Stuff the three resources (INIT, and 2 tpat's) into a file of type INIT, and drop it into your system folder. Reboot and enjoy !

(Actually, this INIT should check for the option key being down before wanting to place a window, that would make it much more useful)

It is tested on a SE/30 with 24bit color and 32bit QD, and on a plain 1meg SE, and both works fine (The SE hasn't Color QD, and thus uses the b/w version)

Happy hacking,

Jon Wätte, Stockholm, Sweden, h+@nada.kth.se

```
/*

    SetWindow.c

    This INIT loads the tpat resource ID 128 for B/W and tpat 129 for color
    systems and patches the ShowWindow trap with that resource.

    Copyright 1990 Jon Wätte. Permission granted to use and distribute if
    you don't charge anything for it. If you do, a quarter of your gross
    sales is mine.
*/


int
strcmp(char * s1, char * s2) /* Since we don't want to link with the ANSI
                             library for just one functino, we do it
                             ourselves. Note, this verision returns 0 on
                             MISmatch ! */
{
    while(*s1 == *s2 && *s2) {
        s1++; s2++;
    }
    if(*s1 == *s2) return 1;
    return 0;
}


main()
{
    char * moof;
    long oldaddr;
    Handle foom;
    int num = 128;
    SysEnvRec theWorld;
```

```
SysEnvirons(2, &theWorld); /* Check what we have here */
if(theWorld.hasColorQD) num++; /* If we have color QD, use the CQD version
                                  which has another number, and supports
```

```
                                        multiple screens */
    foom = GetResource('tpat', num);

    if(foom == 0) { /* Maybe we built the resources with the wrong number ? */
        SysBeep(30); /* Beep to show we didn't load */
    } else {
        HUnlock(foom); /* May be marked as "locked" */
        DetachResource(foom); /* We don't want a resource hanging around in the
                              system heap in that way... */
        MoveHHi(foom); /* Get the code as much out of the way as possible */
        HLock(foom); /* Lock it down firmly, so it won't move... */

        for(moof = *foom; !strcmp(moof, "Moof!"); moof++); /* Check for the
                              availability of our "signature" */

        * (long *) moof = NGetTrapAddress(0x115, ToolTrap); /* Save the address
                              to jump to in place of the signature */
        NSetTrapAddress(StripAddress(*foom), 0x115, ToolTrap); /* Set the new
                              trap address to our routine - note, since the
                              machine possibly might be SwapMMU'ed, we do a
                              StrpAddress - it can't hurt anyway */
    } /* That's it ! Not so hard at all. */
}
```

{---------- Set window tpat ----------}

```
/*
    SetWindow tpat

    This trap patch will make ShowWindow act like X-windows,
    so you may place a new window wherever you like.

    This INIT does lots of stupid things, like pokes in lo-mem globals
    not very well-documented, and draws in the WMgrPort.

    Copyright 1990 Jon Wätte - distribution and usage allowed if you
    don't charge for it. If you do, quarter of your gross sales is mine.

    I'm reachable as Internet: h+@nada.kth.se USEnet: mcsun!sunic!draken!h+

*/

/*
    Things on the to-do list: Maybe turn on/off various features with a
    control panel cdev ? Maybe the mouse should move back again after
    positioning the window ? Maybe we shouldn't beep at dumb applications ?
    Maybe we should show Modal windows without positioning them ?
*/

/* lo-mem globals that are documented, somewhere... */
extern  Point   MTemp           :   0x828;
extern  Point   RawMouse        :   0x82c;
extern  int     CrsrNewCouple   :   0x8ce;

main(WindowPtr w) /* This is the patch. The declaration should look the same
                  as if you were writing the actual routine. Note, that for
                  routines taking more than one argument, pascal declaration
                  is needed. */
{
    char blackPat[8];

    int ofx = w->portRect.right - w->portRect.left,
        ofy = w->portRect.bottom - w->portRect.top; /* Calculate the dimensions
```

```
                                                                of the window */
    int x;
```

```
    asm {
           bra    @done
    moof:  dc     'Mo', 'of', '!\000' /* This is used for communication
                                          with the loader, to see where to
                                          jump next */
    done:  nop
    }

/* Do the stuff here ! */

    for(x=0; x < 8; x++) blackPat[x] = 0x55 << (x & 1); /* Set up a grey
                                                                    pattern */
    if(!(((WindowPeek) w)->visible)) { /* Only if you show a hidden window */

        Point p; /* We have to save away various data to get the thing to
                    work right, and reset the WMgrPort in its state. Otherwise,
                    the various managers would get VERY confured */
        GrafPtr oldPort;
        GrafPtr myPort;
        PenState pnState;
        RgnHandle clipRgn = NewRgn();

        GetPort(&oldPort); /* Whatever port was used - note, usually apps
                              do a SetPort after a ShowWindow, but it never
                              hurts to be nice */
        GetWMgrPort(&myPort); /* This is the port we're gonna draw in */
        SetPort(myPort);

        GetClip(clipRgn); /* We have to change the clip region so we're sure
                             that drawing actually takes place */
        ClipRect(&(myPort->portRect)); /* No way of knowing more than one
                             monitor without Color QuickDraw */

        p = * (Point *) &(w->portBits.bounds); /* Where to move the mouse */
        p.h = -p.h; /* The offsets are negative in bounds ... */
        p.v = -p.v;
         RawMouse = p; /* Hit the mouse lo-mem globals (danger !!!) */
    MTemp = p;
    CrsrNewCouple = 0xffff;

        do {
            long l;
            Delay(2, &l); /* Give the mouse a chance to catch up to the place
                             where the window's default position is */
        } while(Button()); /* See to it the button's up before we go
                              further */

        GetPenState(&pnState);
        HideCursor(); /* We don't want the cursor obscured */
        ShowPen();
        PenSize(2, 2);
        PenMode(patXor); /* Drawing in the WMgrPort requires undo-able
                            ilnes only obtainable by xoring */
        PenPat(blackPat); /* it's really a grey pattern... */

        while(!Button()) {
            Rect r;
            long l;

            GetMouse((Point *) &r);
            r.bottom = r.top + ofy;
            r.right = r.left + ofx;
            FrameRect(&r); /* Show the outline */
```

```
      Delay(2, &l); /* For a short while */
      FrameRect(&r); /* Restore the screen */
   }; /* Until the user clicks */
```

```
      {
         GetMouse(&p); /* Where did the click go down ? */
         LocalToGlobal(&p);
         MoveWindow((WindowPtr) w, (int) p.h, (int) p.v, (Boolean) 0);
            /* Move the window we're placeing there */
      }

      ShowCursor(); /* Restore the saved state of the WMgrPort */
      HidePen();
      SetPenState(&pnState);
      SetClip(clipRgn);
      DisposHandle(clipRgn); /* Don't eat space, either... */

      SetPort(oldPort);
   } else {
      SysBeep(30); /* Here we beep if an application tries to show an
                   already visible window. Good for tracking unnecessary
                   ShowWindows */
   }

   asm {
         move.l @moof, a0 /* The loader looks for "Moof!", and stores
                          the place to jump to there */
         unlk   a6 /* ONLY if you use local variables ! */
         jmp    (a0) /* JMP, not JSR. This is not a tail patch, and thus,
                       shouldn't break any OS patch */
   }
}
```

{ ---------- Set window tpat Color ---------- }

```
/*
   SetWindow tpat

   This trap patch will make ShowWindow act like X-windows,
   so you may place a new window wherever you like.

*/

/* lo-mem globals that are documented, somewhere... */
extern  Point   MTemp           :   0x828;
extern  Point   RawMouse        :   0x82c;
extern  int     CrsrNewCouple   :   0x8ce;

main(CWindowPtr w)
{
   char blackPat[8];
   RGBColor savedC;

   int ofx = w->portRect.right - w->portRect.left,
      ofy = w->portRect.bottom - w->portRect.top;
   int x;

   asm {
         bra    @done
   moof: dc     'Mo', 'of', '!\000'
   done: nop
   }

/* Do the stuff here ! */

   for(x=0; x < 8; x++) blackPat[x] = 0x55 << (x & 1);
```

```
if(!((WindowPeek) w)->visible)) {
    Point p;
```

```
    GrafPtr oldPort;
    GrafPtr myPort;
    RGBColor c = { 0, 0, 0 };
    PenState pnState;
    RgnHandle clipRgn = NewRgn();

    GetPort(&oldPort);
    GetCWMgrPort(&myPort);
    SetPort(myPort);
    GetForeColor(&savedC);

    GetClip(clipRgn);
    SetClip(GetGrayRgn());

    if((w->portVersion & 0xE000) == 0) { /* If an old-style graf port */
       p = * (Point *) &(((GrafPtr) w)->portBits.bounds);
    } else {
       p = * (Point *) &((*(w->portPixMap))->bounds);
    }
    p.h = -p.h;
    p.v = -p.v;
     RawMouse = p;
MTemp = p;
CrsrNewCouple = 0xffff;

    do {
       long l;
       Delay(2, &l);
    } while(Button());

    GetPenState(&pnState);
    HideCursor();
    ShowPen();
    PenSize(2, 2);
    RGBForeColor(&c);
    PenMode(patXor);
    PenPat(blackPat);

    while(!Button()) {
       Rect r;
       long l;

       GetMouse((Point *) &r);
       r.bottom = r.top + ofy;
       r.right = r.left + ofx;
       FrameRect(&r);
       Delay(2, &l);
       FrameRect(&r);
    };

    {
       GetMouse(&p);
       LocalToGlobal(&p);
       MoveWindow((WindowPtr) w, (int) p.h, (int) p.v, (Boolean) 0);
    }

    ShowCursor();
    HidePen();
    SetPenState(&pnState);
    SetClip(clipRgn);
    DisposHandle(clipRgn);
    RGBForeColor(&savedC);
```

```
    SetPort(oldPort);
} else {
    SysBeep(30);
```

```
    }

    asm {
            move.l @moof, a0
            unlk   a6 /* ONLY if you use local variables ! */
            jmp    (a0)
    }
}
```

New Volume Scanning Algorithm

By John Norstad

In Tech Note #68, "Searching All Directories on an HFS Volume", Apple gives a very simple algorithm for disk scanning. There's a problem with this algorithm, however, which I discovered while working on my anti-virus program Disinfectant. I've come up with an improved algorithm that solves the problem. The new algorithm will be part of Disinfectant version 1.1, which we hope to release early next week.

I've wanted to "publish" this new algorithm so that everyone can benefit from it. comp.sys.mac.programmer seems as good a place as any!

Please understand that this problem is not a "bug" in Disinfectant 1.0, despite what MacWeek has to say :-) The "bug" is shared by any program which uses the TN 68 algorithm to do disk scanning, which I suspect is all programs which do disk scanning.

The basic idea outlined in Tech Note #68 is to make indexed calls to the PBGetCatInfo file manager routine. We'll use (abuse) the following notation for these calls:

```
r = PBGetCatInfo(d, i, o)
```

means "call PBGetCatInfo to get the i'th object o in directory d, with result code r." Note that r will be non-zero if there are no more objects in the directory.

The algorithm in TN 68, expressed in pseudo-c and stripped of all the bells and whistles, is as follows:

```
i = 1
while (true) {
   if (PBGetCatInfo(d, i, o)) break
   if o is a subdirectory call ourselves recursively to scan o
   if o is a file scan it
   i++
}
```

This algorithm seems quite simple and fool-proof at first glance, but it only works if you assume that no other users or tasks are creating or deleting files or directories while the scan is in progress.

As an extreme example, suppose we're scanning a server volume that contains two files named A and B and a directory C that contains another 1000 files. Suppose that while we're scanning file B some other user deletes file A. Our index i in the above algorithm is 2 while we're scanning file B. When we finish scanning file B we increment i to 3 and loop, calling PBGetCatInfo to get the third object in the directory. But there are now only two objects in the directory (B and C), so the PBGetCatInfo call returns a non-zero result code and we break out of the loop and quit. The net result is that we end up scanning only 2 out of the 1002 total files on the server!

This problem is most serious when scanning server volumes, where the probability of other users creating or deleting objects is often significant. The problem can also occur on local volumes under MultiFinder if other tasks are creating or deleting objects during a scan, or if our program itself creates or deletes objects on the volume during the scan. (Disinfectant 1.0 suffers from all three problems, but only the server problem is really serious.)

My solution is quite simple. I simply recall PBGetCatInfo immediately after scanning an object to see if it has changed its position in the directory. If the position has changed, I rescan the directory to attempt to locate the new position.

The revised algorithm is:

```
i = 1
while (true) {
```

```
if (PBGetCatInfo(d, i, o)) break
if o is a subdirectory call ourselves recursively to scan o
if o is a file scan it
n = the name of object o
```

```
    if (!PBGetCatInfo(d, i, o)) {  /* recall PBGetCatInfo */
       m = the name of object o
       if (n == m) {               /* usual case - no position change */
          i++                      /* continue scan with next object */
          continue
       }
    }
    oldi = i                      /* save our old location */
    i = 1                         /* start looking for our new location */
    while (true) {
       if (PBGetCatInfo(d, i, o)) {
          i = oldi                /* just in case we've been deleted in
          break                   /* the last few milliseconds */
       }
       m = the name of object o
       if (n == m) {              /* found new location */
          i++                     /* continue scan with next object */
          break
       }
       i++
    }
}
```

There is still an unavoidable window in this algorithm where our PBGetCatInfo indices can get out of synch with reality, but it is now only milliseconds wide instead of seconds or even minutes wide. So the new algorithm is still not perfect, but it's orders of magnitude better than the old naive one.

In my first attempt to design this new algorithm I tried to be fancy - I didn't rescan from the beginning of the directory, but I instead tried to scan backwards or forwards from the current position. This technique was slightly faster, but assumed that the directory was maintained in alphabetical order using the RelString toolbox routine with caseSens=false and diacSens=true. This works OK on normal volumes, but with foreign file systems and in other "non-standard" cases we can't assume that directories are in any particular order. The final algorithm presented above does not depend on directories being maintained in any particular order.

Please note that my new algorithm hasn't yet been put to the acid test of use by millions of real live users. But I think it's reasonable and it has worked just fine in my tests. Apple, of course, knows nothing about all this. If they did they'd probably tell me that it would break in system 7.0 :-) So use it at your own risk, etc., etc.

It's interesting that this problem is not shared by UNIX and other operating systems. In UNIX once an entry is made in a directory its position never changes. When entries are deleted they're simply marked "unused". The system does not attempt to move all the following entries down to close up the hole. There is no attempt made to keep the directories in any particular order.

The new algorithm is part of the reusable module scan.c, which is part of the "public" source code of Disinfectant. Write to me at the address below if you'd like a copy.

Please excuse the length of this posting. I thought this was a nifty trick, and there might be others who will find it useful.

John Norstad
Academic Computing and Network Services
Northwestern University

Bitnet: jln@nuacc
Internet: jln@acns.nwu.edu
AppleLink: a0173
CompuServe: 76666,573