Technical Note TB575
Window Manager Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic--questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&A's can be found on the Macintosh Technical Q&A's web site.

[Nov 01 1993]

Don't call FrontWindow from VBL task

When I call `FrontWindow` from within a VBL task, my system occasionally freezes at this call. Is there any chance that it moves memory?

`FrontWindow` may move memory. It isn't on the list of routines that may move or purge memory, but it should be. It's been patched out since MultiFinder, and shouldn't be called at interrupt time.

You'll have to come up with another method of getting the front window from your VBL task. You may want to keep a shadow copy of your application's window list in your application global area where your VBL task can get to it.

Back to top

How to save and restore window positions

Date Written: 11/16/93

Last reviewed: 11/16/93

How should my application save and restore its windows' positions?

It is very important that you save the positions of windows that you open, so that the next time the user launches your application, the windows will go where they had them last. With data files, the window positions can be stored in either the resource or the data fork.

If you have windows that aren't data windows (i.e., separate files), you should store information about their positions in a separate configuration file. (Storing them as a resource in your application is a bad idea, since they then 1) are not shareable on AppleShare (since resource forks are not) and 2) cannot save their positions if run from a CDROM or other protected volume.)

Under System 7, this file should be put in the Preferences Folder inside the System Folder. Under System 6, this file should be put in the currently open system folder (this is guaranteed to be a local, non-shared volume as opposed to a server volume). See Technote TB535 - Finder Q&As: "FindFolder and Saving Application Preferences", for common code for Systems 6 and 7.

Regardless of where you store window position information, your program should always check that the window's coordinates are reasonable before relocating the window there. That way users with two monitors who rearrange them, or users with large or multiple monitors who give their documents to users with single, small monitors, won't "lose" their windows when your program reopens them somewhere offscreen (for some reason users find offscreen windows hard to read).

It's not enough to know some part of the window is on-screen. There must be title bar pixels on-screen for a user to drag the window (if a window that uses the standard WDEFs is opened with only the lower part of the window showing, there's no way to drag it!). So you need to calculate the title bar's rectangle. Start by finding the coordinates of the window's content region, which is fairly easy. Use the left and right coordinates as-is for the title bar's rectangle; use the top content region coordinate for the title bar's bottom coordinate. For the top of the title bar's rectangle, subtract the title bar height. For help calculating the title bar height (it varies internationally), see the next question, "How to determine Mac window title bar height for localization."

Once you've got the title bar's rectangle, call `InsetRect` to inset it on all sides by four pixels (enough for the user to mouse on to drag the window back onto the screen). Then call QuickDraw's `RectInRgn` to compare the title bar's rectangle against the global variable `GrayRgn`, which is the visible desktop region.

A generalized routine that returns a boolean indicating whether the title bar of the proposed coordinates of a window will be on-screen might look like (in C):

```
// WindowIsOnScreen: Return true if at least 4 pixels of title bar are on-screen
//   Title bar is key, since that's the only window area users can drag it around by

Boolean WindowIsOnScreen(WindowPtr userWindow, const Rect *proposedContentRect)
{
    Rect    dragOnRect;     // calculate rect that user can drag on: the title bar

    dragOnRect.left = proposedContentRect->left;   // get left title bar coord
    dragOnRect.right = proposedContentRect->right; // get right title bar coord
    dragOnRect.bottom = proposedContentRect->top;  // bottom coord = content top

    // now for the top of the title bar: top = bottom - height
    // use GetTitleBarHeight from "How to determine Mac window title bar height..."
    dragOnRect.top = dragOnRect.bottom - GetTitleBarHeight(userWindow);

    // compare title bar against gray region (the visible screen region)
    // first give user 4 pixels to grab
    InsetRect(&dragOnRect, 4, 4);
    return ( RectInRgn(&dragOnRect, GetGrayRgn()) );
```

If `RectInRgn` reveals the window will be visible at the proposed (previous) coordinates, call `MoveWindow` to move it there; otherwise, locate it at some reasonable position--the main screen's upper-left coordinates are at 0,0, but you'll want to position the window below the menu bar by adding both its height (call `GetMBarHeight`) and the title bar's height.

Back to top

How to determine Mac window title bar height for localization

Date Written: 8/12/91

Last reviewed: 11/16/93

How can my application determine a window title bar's height in pixels? Is there a method similar to reading the global `MBarHeight` ($BAA) for the menu bar height?

While the menu bar height can be easily found, determining the title bar height requires some effort. You are right to be concerned about the matter, as international versions of the system software may have various sizes of title bars.

You should also note that Inside Macintosh: Macintosh Toolbox Essentials is incorrect in stating that, in the Roman script system, the standard document title bar is 20 pixels high: it is actually 19 pixels high.

In a nutshell, you need to find the difference between the top of the rectangle of the window's content area, which is below the title bar, and the top of its structure region. Be aware that the window has to be visible for its structure and content regions to be valid. Use `WindowPeek` to get the values. If you know the window is visible, getting the title bar height is simple (in C):

```
  titleBarHeight =
      ( **( ((WindowPeek)userWindow)->contRgn) ).rgnBBox.top
```

Since neither content nor structure region is valid unless the window is visible, you should make sure the window is visible before you examine these regions. The easiest thing to do is to move the window way off-screen, show it (use `ShowHide`, instead of `ShowWindow` and `HideWindow: HideWindow` can change the plane of the window; also, `ShowHide` does not affect the hiliting of windows), calculate the title bar height, then hide it (using `ShowHide` again), and move it back. A generalized routine that returns the window's title bar height, regardless of whether the window is visible, might look like (again in C):

```
// GetTitleBarHeight: Return the height of a window's title bar,
//   regardless of whether the window is visible

unsigned short GetTitleBarHeight(WindowPtr userWindow)
{
    // use kOffscreenLoc from System 7 Samples:DTS.Utilities:Utilities.c
    Point   kOffscreenPoint = {0x4000, 0x4000};
    short   titleBarHeight;                   // determine title bar height
    WindowPeek userWindowPeek = userWindow;   // use to avoid casts

    // to find the top of the title bar, the window must be visible
    // (the strucRgn and contRgn aren't set up if window hasn't been shown
    if (userWindowPeek->visible) {            // if it IS visible
        titleBarHeight =                      // ht = content.top - struct.top
            ( **(userWindowPeek->contRgn) ).rgnBBox.top
               - ( **(userWindowPeek->strucRgn) ).rgnBBox.top;
    }

    // if the window is NOT visible, temporarily move it somewhere it can be...
    else {
        Point   savePoint;        // save global coords of Top-Left of user's window
        GrafPtr savePort;

        GetPort(&savePort);
        SetPort(userWindow);      // set window's port as current port
            // uses GetGlobalTopLeft from DTS.Utilities, System 7 sample code
        savePoint = GetGlobalTopLeft(userWindow);

        // move window way offscrn
        MoveWindow(userWindow, kOffscreenPoint.h, kOffscreenPoint.v, false);
        ShowHide(userWindow, true);  // make it visible (in the twilight zone)
        titleBarHeight =              // ht = content.top - struct.top
            ( **(userWindowPeek->contRgn) ).rgnBBox.top
               - ( **(userWindowPeek->strucRgn) ).rgnBBox.top;
        ShowHide(userWindow, false); // hide it again

        // restore window's previous location
        MoveWindow(userWindow, savePoint.h, savePoint.v, true);
        SetPort(savePort);
    }

    return ( titleBarHeight );
```

For more help, look at DTS.Utilities in the Sample Code folder on the System 7 Golden Master CD, or in the System 7 Samples folder on subsequent Developer CDs (this folder includes a wide variety of other basic, useful routines as well). If you're calculating the title bar height for the purpose of confirming that saved window coordinates are reasonable (before relocating a window to those coordinates), see the previous question, "How to save and restore window positions."

Back to top

Code for implementing a Macintosh grow box but not scroll bars

Date Written: 7/30/91

Last reviewed: 6/14/93

How do I draw a Macintosh grow box without the scroll bars? In the past I avoided having the scroll bars by calling PenSize(-1, -1), which effectively turns off the drawing of the scroll bar lines, and then calling `PenNormal`, but this trick doesn't work in System 7.

There is a very simple way to do this: Change the clipping region of the window before you call `DrawGrowIcon`. Make sure you save the old clipping region or all heck will break loose. Here's a little routine that you can replace calls to `DrawGrowIcon` with:

```
#define    kGrowBoxWidth    15

void MyDrawGrowIcon(WindowPtr window)
{
    GrafPtr savePort;
    RgnHandle saveRgn;
    Rect growRect;

    GetPort(&savePort);
    SetPort(window);

    growRect = window->portRect;
    growRect.top = growRect.bottom - kGrowBoxWidth;
    growRect.left = growRect.right - kGrowBoxWidth;
    saveRgn = NewRgn();
    GetClip(saveRgn);
    ClipRect(&growRect);
    DrawGrowIcon(window);
    SetClip(saveRgn);
    DisposeRgn(saveRgn);
    SetPort(savePort);

    DrawGrowIcon(window);
```

Just paste this code into your code and replace all calls to `DrawGrowIcon` with `MyDrawGrowIcon`.

Back to top

Macintosh tool palette windoid reference

Date Written: 7/30/91

Last reviewed: 8/1/91

Do you have any sample code or WDEFs for Macintosh tool palette windoids?

Floating windows or windoids are not supported by Apple's system software. The best example has been published in the MacTutor article of April and May in 1988. (It's reprinted in The Definitive MacTutor, Volume 4, entitled "Tool Window Manager.") Its limitation is that it will only support a single window. There are also problem when using color QuickDraw regarding the Palette Manager, and we have not seen a good solution.

Unfortunately, no standard WDEF has been issued by Apple, nor have standards been set for appearance and characteristics of floating windows. Those may be available in the future, but we presently have no guidelines.

Back to top

Implementing Macintosh floating windows or palettes

Date Written: 8/15/91

Last reviewed: 8/15/91

We're having a lot of trouble with our floating palette which uses the global variable `GhostWindow` documented in *Inside Macintosh* Volume I. What's the best way to implement a floating palette?

There is no easy, built in, and supported way to do floating windows/palettes on the Macintosh. None of the solutions available are particularly elegant, but they do work.

Using the `GhostWindow` low-memory global to do floating windows is not a good idea, mostly because it's a low-memory global, but also because it only allows one "floating" window in an application.

A solution that should work well for you, but it is not incredibly elegant, is to write "wrapper" functions for the Window Manager functions. Basically, your application doesn't call many of the Window Manager functions (`BringToFront/FrontWindow/MoveWindow`) but instead calls routines that keep track of where the floating windows are, and how to keep them in front.

MacTutor has had a couple of good articles on this topic:

- "Tear-off Menus & Floating Palettes," by Don Melton and Mike Ritter (MacTutor 4:4), describes how to do tear-off menus and turn them into floating palettes.
- "Tool Window Manager," by Thomas Fruin (MacTutor 4:12), gives source (in C) to a new "window manager" which handles floaters the correct way through wrapper functions.

Back to top

How the system WDEF determines color

Date Written: 10/23/91

Last reviewed: 2/17/92

We're using `wctb` 0 and `cctb` 0 of the system so our WDEF and `CDEF` will be 7.0 friendly. The Macintosh Technical Note "Color, Windows and 7.0" specifies the use of all the `wctb` table components, but in cases such as the title bar background and scroll bars, it states that colors are generated algorithmically without specifying the algorithm.What is the algorithm? Is it safe to assume that it's 50% of the light and dark components of the resource?

A more recent version of the Technote describes the way the system WDEF determines color. The trick is to use `GetGray` to determine the color from the two extremes available in the color table for the monitor on which the window is going to appear. `GetGray` is described in the Color QuickDraw section of *Inside Macintosh* Volume VI. The latest *Developer CD Series* disc has the system WDEF, which could help you in your work. The path to the file is Dev CD VIII: Tools & Apps (Moof!):OS/Toolbox: System 7 WDEF.

Back to top

Detaching a WDEF from its resource file

Date Written: 12/10/91

Last reviewed: 6/14/93

We sometimes need to close a resource file while still keeping a window open which is using a WDEF loaded from that resource file. We have been detaching the WDEF resource, which means that the WDEF handle in the window record is no longer a resource handle. What problems are we likely to have with this solution?

Your solution is quite adequate. After a window is created, the system has no further need to use resource-manipulation calls on its window definition. As long as the handle is nonpurgeable, there should be no need to reload the data, so the Toolbox should be quite content with that state of affairs.

Back to top

Custom WDEFs in DAs not possible

Date Written: 3/30/92

Last reviewed: 5/21/92

I am writing a DA that uses a custom WDEF. Is there any way to get a window to use a WDEF that is an owned resource? I tried giving the WDEF ID -16000 (DRVR ID = 12, base ID 0) and using `SetResInfo` to change the WDEF's ID. Unfortunately, this only works the first time I run the DA. I have also tried to use `AddResource`. This invariably fails with `ResError` = -194. Can you suggest any way to get this to work, other than by using an Installer to put the WDEF into the user's System file when installation the application?

The owned resource mechanism and the WDEF numbering scheme conflicts in such a way that it's not possible to include custom WDEFs in DAs, since the DA mover will not move them unless they are in the owned resource range and the WDEF numbering scheme does not permit this. There's basically no workaround to this. Installing your WDEF in the system directly could cause numbering conflicts with additional WDEFs in future systems or other DAs doing the same thing. This is the reason for having owned resources.

Back to top

Macintosh DA with custom 'WDEF'

Date Written: 7/24/90

Last reviewed: 6/14/93

Is it acceptable to write an installation program for a DA that uses a custom `'WDEF'`? The Font/DA Mover won't install a `'WDEF'` because the `'WDEF'` cannot be an owned resource.

Although the owned resource mechanism and the `'WDEF'` ID with variation code are directly incompatible with each other, it is a simple matter to work around this problem with just a little code.

First, number your `'WDEF'` resource as an owned resource, following the instructions on page 109 of *Inside Macintosh* Volume I. This will allow your users to simply use the DA/Font Mover to install/deinstall your DA and eliminate the need for you to do a custom installer with all the problems that involves.

Next, when you open a window that needs your custom `'WDEF'`, open it "invisible" and with a standard built-in `'WDEF'` procedure ID; GetResource the `'WDEF'` resource handle yourself and store the handle into the windowDefProc field of the window record. Its "owned" resource ID is calculated from the DA's installed driver unit number, dCtlRefNum. Note: a refNum of -4 is a unit number of 3. See *Inside Macintosh* Volume II, page 191, for details.

Finally, show your window and it should be drawn using the custom `'WDEF'`.

Back to top

Downloadables

Acrobat version of this Note (K). Download

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code