# New Technical Notes
Macintosh

Developer Support

## Window Manager Q&As
**Toolbox**

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

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

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

---

### How to determine Mac window title bar height for localization
Date Written: 8/12/91
Last reviewed: 8/13/91

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.

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 rect to be valid. Given those, you can make your decision on where to place the window.

To get a better idea on how to implement these, take a look at the procedures pasted below. They are taken from DTS.Utilities in the Sample Code folder on the System 7 Golden Master CD (which includes a wide variety of other basic, useful routines as well). Or more directly, here’s an excerpt based on the DoWZoom routine in the Macintosh Technical Note "_ZoomWindow" which includes the title bar height in its calculations:

```
 GetPort(savePort);
 SetPort(window);

 windRect :=window^.portrect;

 LocalToGlobal(windRect.topLeft);
 LocalToGlobal(windRect.botRight);
 titleHeight := windRect.top - WindowPeek(window)^.strucRgn^^.rgnBBox.top;

 SetPort(savePort);
___

{ excerpted from DTS.Utilities:Utilities.inc1.p }

FUNCTION    CenterWindow(window:WindowPtr; relatedWindow:WindowPtr): Rect;

{ Center a window within a particular device. The device to center the window within is
determined by passing a related window. This allows related windows to be kept on the same
device. This is useful if an alert related to a specific window, for example. }

 VAR
   whichDevice:    WindowPtr;
   deviceRect, oldWindowRect, newWindowRect, contentRect:    Rect;

 BEGIN
   whichDevice := relatedWindow;
   if whichDevice = NIL THEN
     whichDevice := window;
       { If we have a window to center against, use the device for
         that window, else use the device for the window that is
         getting centered. }

   deviceRect := GetWindowDeviceRectNMB(whichDevice);
     { We now have the rectangle of the device we want to center within. }

   newWindowRect := GetWindowStructureRect(window);
   oldWindowRect := newWindowRect;

   PositionRectInRect(deviceRect, newWindowRect, FixRatio (1, 2),
                                                 FixRatio (1, 3));
     { Figure out the new window strucRect so we can compare it against
       the old strucRect. This will tell us how much to move the window  }

   contentRect := GetWindowContentRect(window);
     { Get where the window is now. }

   OffsetRect(contentRect, newWindowRect.left - oldWindowRect.left,
                           newWindowRect.top - oldWindowRect.top);
     { Calculate the new content rect. }

   MoveWindow(window, contentRect.left, contentRect.top, false);
     { Move the window to the new location. }

   CenterWindow := contentRect;
 END;

FUNCTION    GetWindowContentRect(window:WindowPtr):Rect;

{ Given a window pointer, return the global rectangle that encloses the
 content area of the window. }

 VAR
```

```
   oldPort:        WindowPtr;
   contentRect:    Rect;

 BEGIN
   GetPort(oldPort);
   SetPort(window);
   contentRect := window^.portRect;
   LocalToGlobalRect(contentRect);
   SetPort(oldPort);
   GetWindowContentRect := contentRect;
 END;

FUNCTION    GetWindowStructureRect(window:WindowPtr):Rect;

{ This procedure is used to get the rectangle that surrounds the entire structure of a
window. This is true whether or not the window is visible. If the window is visible, then
it is a simple matter of using the bounding rectangle of the structure region. If the
window is invisible, then the strucRgn is not correct. To make it correct, then window has
to be moved way off the screen and then made visible. This generates a valid strucRgn,
although it is valid for the position that is way off the screen. It still needs to be
offset back into the original position. Once the bounding rectangle for the strucRgn is
obtained, the window can then be hidden again and moved back to its correct location. Note
that ShowHide is used, instead of ShowWindow and HideWindow. HideWindow can change the
plane of the window. Also, ShowHide does not affect the hiliting of windows. }

 VAR
   oldPort:        GrafPtr;
   structureRect:    Rect;
   windowLoc:        Point;

 BEGIN
   if WindowPeek(window)^.visible = true THEN
     structureRect := WindowPeek(window)^.strucRgn^^.rgnBBox
   ELSE BEGIN
     GetPort(oldPort);
     SetPort(window);
     windowLoc := GetGlobalTopLeft(window);
     MoveWindow(window, windowLoc.h, kOffscreenLoc, false);
     ShowHide(window, true);
     structureRect := WindowPeek(window)^.strucRgn^^.rgnBBox;
     ShowHide(window, false);
     MoveWindow(window, windowLoc.h, windowLoc.v, false);
     SetPort(oldPort);
     OffsetRect(structureRect, 0, windowLoc.v - kOffscreenLoc);
   END;

   GetWindowStructureRect := structureRect;
 END;
```

## Code for implementing a Macintosh grow box but not scroll bars

Date Written: 7/30/91
Last reviewed: 8/12/91

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.

## 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.

## 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.

## 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 Tech Note 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.

## Detaching a WDEF from its resource file

Date Written: 12/10/91

Last reviewed: 1/27/92

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.

**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.

**Macintosh DA with custom 'WDEF'**
Date Written: 7/24/90
Last reviewed: 10/1/91

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'.