New Technical Notes
Macintosh

®

Developer Support

Palette Manager Q&As
Imaging M.IM.PaletteMgr.Q&As

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

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

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

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

Macintosh Palette Manager and offscreen graphics
Written: 7/22/91
Last reviewed: 8/1/92

The Macintosh Palette Manager doesn't work on offscreen environments the way you'd expect. Unlike color windows, SetPalette will not change the offscreen's color table; rather it just allows you to use PMForeColor and PMBackColor to set the current drawing color in that environment. To change the offscreen's color table you'll need to convert the palette to a color table and then set the resulting color table to the off screen. Calling Palette2CTab will do the converting for you.

To get around having to use palettes to define the current drawing color in the off screen, you can always use Index2Color and then RGBForeColor to get the color to be set for drawing. A remake of GiMeDaPalette code sample, available on the latest Developer CD Series disc, does offscreen drawing in place of having to continually copy a PICT.

RestoreDeviceClut and color flash when application quits

Written: 8/23/91
Last reviewed: 8/1/92

When my application, which uses a color palette, quits, there is momentary but distracting flash of weird colors in the Finder windows and the desktop temporarily appears in a weird color. Is there any way to get around this?

___

When you quit, RestoreDeviceClut is called to restore the color table and an update event is called to redraw the screen. It's the delay between the change in the color table and the update event that causes the flash of incorrect colors to be displayed. This, unfortunately, is unavoidable.

Restoring Finder desktop colors after using Palette Manager
Written: 8/28/91
Last reviewed: 8/1/92

After using the Macintosh Palette Manager, how do I restore the Finder's desktop colors?

___

The Finder desktop's colors are restored automatically on quitting applications that use the Palette Manager. Colors aren't restored automatically when switching from your application to another, but if that application needs a certain set of colors and uses the Palette Manager to get them, then it'll have them the moment it comes to the front. If you're concerned about applications that don't use the Palette Manager, you can use RestoreDeviceClut(gd:GDHandle), passing the handle to the GDevice of the screen you want to reset, or nil if you want to reset all of your devices. Passing nil to RestoreDeviceClut is your best bet, as it is very straightforward, and resets all of your monitors. You may not wish to do this, however, because RestoreDeviceClut is only available on machines with 32-bit color QuickDraw.

To reset a screen's GDevice for machines without 32-bit color QuickDraw, you will need to keep track of the color table.When your application starts up, get the GDevice's color table and save it—you'll need it later. This value can be found at (**(**GDHandle).gdPMap).pmTable, where gdPMap is a PixMapHandle, and pmTable is a CTabHandle which tells you the absolute colors for this image. These data structures are found in Inside Macintosh Volume V, pages 52 and 119.

Build your application's "world" using the Palette Manager, and avoid low-level methods of changing colors. When your application is about to quit and you want to restore the environment to its original state, get the color table you saved in the beginning. Convert this to a palette using CTab2Palette. Then set your window to this palette with SetPalette. This will cause the environment to update to the original color table that you initially got from the GDevice. If the application that is behind your application is Palette Manager friendly, then it will restore the environment to its palette. You may also want to do this procedure at the suspend event, as shown in the DTS sample MacApp program, FracApp. One of the problems that you won't be able to solve this way involves multiple monitors. You won't know which one to update. Only the monitor that has the window that you've called ActivatePalette on will update.

If your application changes the color environment with the Palette Manager, then RestoreDeviceClut is called automatically when your application quits. This means that you shouldn’t have to worry about restoring the palette if you don’t want to. There is a catch, however (there always is). When you use the SADE version of MultiFinder (6.1b9), it prevents this from automatically happening. Other versions of MultiFinder don’t have this side effect.

SetPalette cUpdates, NSetPalette, and window update events

Written: 9/26/91

Last reviewed: 8/1/92

When I pass false in the cUpdates parameter to SetPalette, I still get update events to that window when I modify its palette. What's going on?

___

SetPalette's cUpdates parameter controls whether color-table changes cause that window to get update events only if that window is NOT frontmost. If that window is frontmost, then any changes to its palette causes it to get an update event regardless of what the cUpdates parameter is. When you call SetEntryColor and then ActivatePalette for your front-most window, it gets an update event because it's the front-most window even though you passed false in the cUpdates parameter. Another important point is that windows that don't have palettes also get update events even when another window's palette is activated.

Fortunately, system software version 6.0.2 introduced the NSetPalette routine, which you can find documented in the Macintosh Technical Note "Palette Manager Changes in System 6.0.2," and on page 20-20 in the revamped Palette Manager chapter of Inside Macintosh Volume VI. This variation of SetPalette gives you a lot more flexibility in controlling whether your window gets an update event than SetPalette does. If you pass pmAllUpdates in the nCUpdates parameter, then that window gets an update event when either it or another window's palette is activated. If you pass pmFgUpdates, then it gets an update event when a palette is activated only if it's the front-most window (in effect, it only gets an update event if its own palette is activated). If you pass pmBkUpdates, then it gets an update event when a palette is activated only if it's not the front-most window (in effect, it only gets an update event if another window's palette is activated). If you pass pmNoUpdates, then that window never gets an update event from any window's palette being activated, including that window itself.