Technical Note QD530

Palette 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&As can be found on the Macintosh Technical Q&As web site.

[Oct 01 1990]

GetNewPalette and default palette documentation update

Date Written: 3/10/93

Last reviewed: 6/24/93

Our application has a `'pltt'` resource containing colors, which is replaced by the user's palette from a Preferences file if it exists. Sometimes our Color dialog box displays black-and-white or incorrect colors. Does the Palette Manager documentation describe default palettes correctly? Should I call `GetNewPalette(0)` on my application resource fork instead of calling `GetPalette(-1)`?

The actual implementation of the Palette Manager is somewhat different than documented in Inside Macintosh Volume VI.

The Palette Manager has two levels of default palettes. First, there's a system palette that's loaded at system startup time. This system palette (`'pltt'`, ID#0) is loaded from the system file if it exists and stored in the system heap along with other Palette Manager globals. Once the system is booted, this system palette doesn't change. Even if you replace the palette in the system file, the new palette isn't reloaded. If there's no system palette resource in the system file, the Palette Manager hard codes a 2 entry (black and white) palette as the system palette. The default system file for both Systems 6.0.8 and 7.1 doesn't have a (`'pltt'`, ID#0) resource. Therefore, it seems that the default system palette is typically an entry palette with black and white.

The other level of default palette is the application palette. This palette is stored in a low-memory global called `AppPalette`. You probably won't find it in any documentation, but the address is $DCC. (This low-memory global is switched during context switches so you don't have to worry about losing the low-memory global when switching applications.) The Palette Manager loads this default application palette when `InitPalette` is called. Normally, you don't call `InitPalette` directly in your code. Instead, `InitWindows` calls `InitPalette` for you. Inside `InitPalette`, a call to `Get1Resource` is made with (`'pltt'`, ID#0). `Get1Resource` differs from `GetResource` because `Get1Resource` checks only the first resource file. Since your application is assumed to be at the top of the resource chain, this call will get a palette of ID #0 if it's available in that file *only*. This actually causes a problem when running an application from a development environment such as Think C.

Unlike the default system palette, the default application palette can be modified by a call to `SetPalette(WindowPtr(-1), newDefaultAppPalette`, TRUE). You can also get the default palette with a call to `GetPalette(WindowPtr(-1))`. When there's no application default palette, `GetPalette(WindowPtr(-1))` uses the default system palette instead. As mentioned earlier, this is typically a 2 entry palette with black and white.

Lastly, the documentation for the call GetNewPalette on page 20-19 of Inside Macintosh Volume VI is completely misleading. GetNewPalette simply loads in the specified 'pltt' resource using GetResource and then detaches it with DetachResource to make it a handle. It doesn't attach it to the current window. And, it doesn't load the default application handle if the specified resource can't be found.

Back to top

SetPalette cUpdates, NSetPalette, and window update events

Date Written: 9/26/91

Last reviewed: 6/14/93

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 the frontmost window. If that window is the frontmost window, any changes to its palette cause it to get an update event regardless of what the cUpdates parameter is. When you call SetEntryColor and then ActivatePalette for your frontmost window, the window gets an update event because it's the frontmost window even though you passed FALSE in the cUpdates parameter. Another important point is that windows that don't have palettes always get update events when another window's palette is activated.

Fortunately, system software version 6.0.2 introduced the NSetPalette routine, which is documented in the Macintosh Technical Note "Palette Manager Changes in System 6.0.2" and on page 20-20 in the Palette Manager chapter of *Inside Macintosh* Volume VI. This variation of SetPalette gives you the following options in controlling whether your window gets an update event:

- If you pass pmAllUpdates in the nCUpdates parameter, your window gets an update event when either it or another window's palette is activated.
- If you pass pmFgUpdates, your window gets an update event when a palette is activated only if it's the frontmost window (in effect, it gets an update event only if its own palette is activated).
- If you pass pmBkUpdates, your window gets an update event when a palette is activated only if it's not the frontmost window (in effect, it gets an update event only if another window's palette is activated).
- If you pass pmNoUpdates, your window never gets an update event from any window's palette being activated, including that window itself.

Back to top

Restoring Finder desktop colors after using Palette Manager

Date 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.

Back to top

RestoreDeviceClut and color flash when application quits

Date Written: 8/23/91

Last reviewed: 6/14/93

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.

Back to top

Macintosh Palette Manager and offscreen graphics

Date 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.

Back to top

Downloadables

Acrobat version of this Note (K). Download