About the Palette Manager Information on additional support for Color QuickDraw

This topic covers routines that help your applications manage shared colors; how to match colors precisely; and introduces you to animation concepts.

The **Palette Manager** handles the details of which window gets what when applications are contending for color resources, assigning priority and assuming a first-place position for the front window. It becomes active the first time you call InitWindows and thereafter runs whenever windows are moved. The **Palette Manager** uses a default palette to provide a color environment for windows that haven't been given a palette of their own--including pre-**Color QuickDraw** windows.

A modification of the default palette for System 6.0.2 is called the **application palette**, which lets the System define the color environment when it either creates a color window that doesn't have its own palette or when it displays a dialog box. When in place, the application palette supersedes the default palette when a color application uses old-style dialog boxes, avoiding the problem of colors from the default palette showing up to change the color environment. In these cases, defining an applications palette with black and white as the sole foreground and background choices solves the problem.

While the default palette is a convenience, it only goes so far in providing your applications with the full range of colors available from the Mac II-family. This is where the **Palette Manager** comes in. With it, you get a means to revise a window's color environment automatically--thus giving your applications access to a wide selection of shades, both in color and in an extensive gradation of grays.

You start like this. Create a data structure called a color **palette**, either via the resource type **'pltt'** or through a **Palette Manager** routine. Within this data structure you include a palette entriy for each a color you want your application to have. Additionally, the entries contain information that tells the **Palette Manager** you want the colors to match exactly. You use a built-in routine to tie your newly-created palette to a specific window. Whenever you bring that window to the front, the **Palette Manager** checks to see if the color environment suits the window. If not, the **Palette Manager** borrows any extra colors it needs from background windows.

Another couple of **Palette Manager** capabilities are worth mentioning here. One is that it relieves your application of trying to keep track of multiple machine configurations since it automatically handles multiple devices and a variety of screen resolutions. The second is that it builds a color index model on top of the **Color QuickDraw** RGB model -- thereby providing such features as special animation effects.

Color Indexing

Color indexing is a system whereby RGB values are assigned specific positions in a video card's color lookup table and, generally, the video card has far more colors available to it than just those listed in the index. This is both good and bad. The good part is that memory requirements are low since only the index, and not the whole color table, is in RAM at any one time. Lower memory requirements translate into higher speed since a proportionally smaller amount of data is written to the screen, as compared with a direct color model

as explained in **Color Manager**. It also means that if you change index entries instead of pixel values when changing colors, all pixel values representing the changed index entries automatically show up on the screen in the their new colors. Thus, by changing index values you can provide an application with realtime movement, or animation.

The primary limitation of the color indexing model is that it only gives you a relatively small number of colors to work with at any one time. All of the colors for all of the applications sharing a window have to come from the same index.

The way the **Palette Manager** balances the positive and negative aspects of color indexing is to build its specific model on the more general RGB model. By building palette entries from RGB values, this new Toolbox routine can figure out the best way to take advantage of a particular monitor's color capabilities.

Color Selection

The **Palette Manager** uses four color classifications to help decide just how to handle a color request from an application. It calls colors: courteous, tolerant, animating or explicit, depending on how and where a specific application uses specific colors.

Courteous colors (pmCourteous, specified by the usage constant 0x0000) are those without special properties, acting as placeholders; and reserving them against use by animation procedures whenever possible.

Tolerant colors (pmTolerant, specified by the usage constant 0x0002) give you a chance to change the current color environment if you have to. If you specify a tolerant color and the difference between it and the closest possible match is greater than the tolerance level you specify, the **Palette Manager** changes the color environment to the extent that it provides an exact match. What this tells you, though, is that if the available color is within the tolerance range, you'll get the best color the **Palette Manager** thinks you ought to have--which may or may not be the exact shade you specified. A tolerance value of 0x0000 for any color means only an exact match is acceptable while a value of 0x5000 is considered adequate for most color matching without causing undue updating.

Animating colors (pmAnimated, specified by the usage constant 0x0004) let you reserve specific device indexes for color table animation. The **Palette Manager** begins by checking to see what colors are being used by each window and then takes the least-used colors and reserves them to your animating palette. Once there, they can't be called using RGBForeColor. The **Palette Manager** then changes the color environment to match your palette's specified RGB values. The AnimateEntry and AnimatePalette calls activate color table animation--as long as you've already used the color in a drawing by calling PmForeColor or PmBackColor.

An **Explicit** color designation (pmExplicit, specified by the usage constant 0x0008) causes the **Palette Manager** to ignore a color's RGB value within a palette. Explicit colors make no changes to the color environment and are ignored by animation calls. They always match the device index and are used

primarily when you want to give yourself an indexed color interface. You do this simply by setting your window's palette to contain as many explicit colors as are in the screen that has the most indexes. You then use PmForeColor to configure the color graphics port to draw with any index you choose.

As a sidelight, explicit colors let you monitor a particular color environment. You can, for instance, draw a 16 by 16 grid of 256 explicit colors and the colors shown would exactly match those in the device's color table. If you showed this grid in a small window during an animation procedure, the colors it shows would change as the animation progresses.

Who Chooses First?

With all of the possible ways colors can be assigned, a system of establishing priority needs to be in control. In the first place, only **tolerant** and **animating** colors are subjected to contention since **explicit** and **courteous** colors are important only when your application calls PmForeColor or PmBackColor or when it checks all of the palettes for total color usage. However, when your application calls the ActivatePalette routine on bringing a window to the front or after a change to a palette's color or usage values, the **Palette Manager** runs through a cross-check to make sure that the **animating** color requirements are satisfied first. After it goes through all of the available indexes on behalf of the animating colors, all of the **tolerant** colors are given their own unique indexes. If a tolerant color has to make do with a value that's beyond the range you specified the device entry is marked for change to the specified color. If changes are needed in the overall color environment, the **Palette Manager** tells the **Color Manager** to do so--and then goes around and fixes any other windows that have been been affected by the first series of changes.