About the Color Manager Information on color support for Color QuickDraw

This topic covers how to apply routines that help your applications fine-tune color-matching procedures. You'll also be shown utility functions that'll increase your understanding of how the system operates but which you'll rarely have occasion to use.

Color Manager is an intermediary step between high-level programs like **Color QuickDraw** and the lower-level routines that comprise a gDevice.

We're advised that the normal procedure for fine-tuning color application is to let the high-level software handle it. That having been said, **Color Manager** works best with hardware that deals with coloration through a 'clut' (or Color Loop-up Table, see **Color QuickDraw**) resource type that maps an indexed color into an actual color value. While **ColorQuickDraw** will work to some degree with **fixed devices** (ones that display an unchangeable palette of colors but still convert a pixel value into an RGB video value), it is not designed to support **direct devices** that directly translate frame buffer values into screen colors.

Color Manager works with **Color QuickDraw** to scan the lookup table and come up with the best color matches that the color table contains. When your applications need more control, such as for animation, the **Palette Manager** lets them take over a part of the color table for their own use. Right now, it's useful to remember that **Color Manager** operates on the device level, while **Palette Manager** is designed for use on objects within windows.

The tie-in between **Color Manager** and **Color QuickDraw** is that they both access monitors, printers and offscreen drawing areas through the gDevice record structure. (The gDevice record in turn, is accessed through the gdHandle.) The gDevice record is where all of the information pertaining to a particular graphics device is stored. After the record is initialized, the graphics device itself is thereafter referred to as the gDevice. It remains a logical device, however, and the software treats it the same way no matter if it's a video card, monitor, printer or offscreen drawing area.

The format of ColorTable, ColorSpec and RGBColor record structures can be found in the section on **Color QuickDraw**. That information, combined with the following data on **inverse tables** will take you through an understanding of how **Color QuickDraw** specifies colors and gets them onto the screen.

An **inverse table** is a special data structure arranged so that, given any RGB color, its pixel value can be found. It was devised because, on 'clut' or fixed graphics displays there won't always be a direct correlation between **Color QuickDraw**'s absolute RGB specifications and what's available in the hardware. This is the point at which **Color Manager** is called in, and where the **inverse table** has a part to play.

Color Manager extracts and concatenates a group of the most significant bits of the red, green and blue components in the requested absolute color. This, then, becomes the closest available match to the specified RGB. The 3, 4 or 5 most significant bits used to create this index is called the **resolution** of the inverse table. The greater the number of most significant bits, the higher the resolution--and the greater the number of color choices available. Also,

the tables themselves grow as the number of most significant bits grows. A 3-bit table takes up 512 bytes, 4-bit tables need 4K bytes and 5-bit tables will take up 32K bytes in the application heap.

The inverse table method of color matching has limitations when it comes to finding colors that are different in their less significant bits. The way around that is to load the inverse tables with extra information to help your application find colors whose differences are described by these less significant bits. The **Color2Index** function was designed to find the best match to the complete 48-bit resolution of a colorSpec, but the tradeoff is time. So much time, in fact, that such features of **Color QuickDraw** as drawing in the arithmetic transfer modes, are designed not to use this information at all.

Modifying a color table means having to rebuild the inverse table and redrawing the screen to take the new information into account. However, reconstructing an inverse table doesn't happen automatically when the color table is changed. The inverse table is simply marked as invalid and the next time it is accessed is when it is rebuilt.

To determine whether or not an inverse table needs to be rebuilt, color-matching code compares a given ColorTable's ctSeed with its corresponding inverse table's iTabSeed. If the two seeds don't match (and they won't if any routine except **RestoreEntries** was applied), the table is rebuilt.

If necessary, performing a color table look-up is taken in three steps. First, **Color Manager** builds a table from all possible RGB values. Next, it attempts to find the closest match for each indexed position, and finally, it reduces the resolution to 4 bits (the inverse table's default resolution) during table construction (but leaves the door open for adding information later to get better resolution).

Color Manager goes through this exercise whenever **Color QuickDraw**, **Color Picker**, or **Palette Manager** request a color.