

Colors and Color-Related Objects

This chapter describes the QuickDraw GX color architecture and the objects and structures with which you manipulate colors. Read this chapter if your application does any color drawing or calculation, or if you create or modify bitmaps or color sets. Read this chapter also if you are creating a calibration program to generate color profiles.

Before reading this chapter, you should be familiar with the information in the chapter “Introduction to QuickDraw GX” in this book. You should also be familiar with shape objects, as discussed in the chapter “Shape Objects” in this book.

This chapter constitutes the complete discussion of color for QuickDraw GX. Unlike for shape objects and style objects, there is no additional discussion of color-related objects in other books. However, additional information relevant to color is in the chapter “Ink Objects” in this book.

QuickDraw GX uses color-matching methods provided by the Macintosh ColorSync Utilities. For information on ColorSync, its color-matching capabilities, and the structure of the color profiles it uses, see the ColorSync chapter of *Inside Macintosh: Advanced Color Imaging* and the Component Manager chapter of *Inside Macintosh: More Macintosh Toolbox*.

For general information on color theory and color spaces, you may also want to read other books such as these: *Measuring Color*, by R.W.G. Hunt, John Wiley & Sons, New York, 1991; *Illumination and Color in Computer Generated Imagery*, by Roy Hall, Springer-Verlag, New York, 1989; and *Computer Graphics: Principles and Practice*, by J. Foley, A. van Dam, S. Feiner, and J. Hughes, Addison-Wesley, Reading, 1990.

This chapter introduces how color is represented in QuickDraw GX, and describes color set objects and color profile objects and their properties. It then shows how to use QuickDraw GX functions to

- assign colors to shapes
- compare, test, and convert colors
- automatically use the color-matching capabilities of QuickDraw GX
- create and manipulate color profiles for imaging devices
- manipulate the colors of a bitmap that uses indexed colors

About Color in QuickDraw GX

In QuickDraw GX, color information about a shape is kept in the ink object associated with the shape object. A shape’s ink object describes both the color of the shape and the transfer mode with which the shape is drawn. Ink objects are described in the chapter “Ink Objects” in this book; colors are described in this chapter.

QuickDraw GX has a powerful, device-independent method for representing color in many different formats. Conversion among the formats is simple and direct, and in many cases automatic. QuickDraw GX also provides automatic manipulation of device-specific colors so that colors match consistently when scanned from or drawn to many different imaging devices.

Colors and Color-Related Objects

This section describes how color is represented and how you can manipulate color information. It presents the information in this order:

- Colors are numerical values that make sense only in terms of specific color spaces. Color spaces are described first, under “Color Spaces” (next section).
- The mathematical values used by each color space are combined with other information to make a color structure. How color values relate to the color structure is described second, under “Color-Component Values, Color Values, and Colors” beginning on page 4-25.
- Colors in a given color space or produced with a given input or display device commonly must be converted to another color space or matched to the color capabilities of another device. How QuickDraw GX accomplishes that task is described third, under “Color Conversion and Color Matching” beginning on page 4-26.

Color Spaces

A *color space* specifies how color information is represented. It defines a one-, two-, three-, or four-dimensional space whose dimensions, or *components*, represent intensity values. For example, RGB space is a three-dimensional color space whose components are the red, green, and blue intensities that make up a given color. Visually, these spaces are often represented by various solid shapes, such as cubes, cones, or polyhedra. See, for example, Color Plate 4 at the front of this book.

QuickDraw GX directly supports 28 different color spaces, to give you the convenience of working in whatever kinds of color data most suits your needs. The QuickDraw GX color spaces fall into several groups, or *base families*. They are

- luminance-based color spaces, used for grayscale display and printing
- RGB-based color spaces, used mainly for color video display
- CMYK-based color spaces, used mainly for color printing
- universal color spaces, used mainly for device-independent color measurements

All color spaces within a base family differ only in details of storage format or else are related to each other by very simple mathematical formulas. Conversion of color across base families is more complex, as described in the section “Color Conversion and Color Matching” beginning on page 4-26.

Within a base family, some of the differences among color spaces relate to their *packing*, the number of bits used to store each color component. For example, RGB colors might be stored with 5, 8, or 16 bits per component. Each storage format is a different color space. Internally, QuickDraw GX always converts colors so that each component has 16 bits; thus you can think of the 16-bit-per-component color spaces as the fundamental ones in each base family, and those with smaller storage spaces as packed (storage-compressed) versions.

Colors and Color-Related Objects

Some QuickDraw GX color spaces have an alpha channel, an additional component that measures opacity or transparency. Alpha channels are described in the section “Color Spaces With Alpha Channels” beginning on page 4-24.

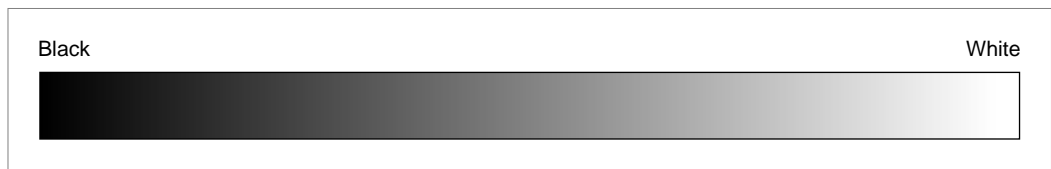
QuickDraw GX also supports a derived color space—indexed color space—in which colors are indirectly specified, using values that are indexed positions in a list. The colors in that list, however, must still belong to one of the base-family color spaces.

The `gxColorSpaces` enumeration, shown on page 4-55, lists the color spaces directly supported by QuickDraw GX. Each color space has its own format for representing color information. The rest of this section discusses those color spaces and their formats.

Luminance-Based Color Spaces

Luminance is a scale of lightness. Luminance-based color spaces, or gray spaces, typically have a single component, ranging from black to white, as shown in Figure 4-1. Luminance-based color spaces are used for black-and-white and grayscale display and printing.

Figure 4-1 Luminance color space



A color is converted into luminance by evaluating its overall lightness. The luminance of a color expressed in RGB (see “RGB-Based Color Spaces” beginning on page 4-9), for example, can be calculated approximately with this formula:

$$\text{luminance} = 0.30 * \text{red} + 0.59 * \text{green} + 0.11 * \text{blue};$$

(QuickDraw GX provides a function for converting colors among different color spaces.)

The luminance-based color spaces supported by QuickDraw GX (and defined in the `gxColorSpaces` enumeration) are `gxGraySpace` and `gxGrayASpace`. The *A* in `gxGrayASpace` stands for a second component called an *alpha channel*; see the section “Color Spaces With Alpha Channels” beginning on page 4-24 for more information.

Table 4-1 describes details of the storage formats for `gxGraySpace` and `gxGrayASpace`. In each of these spaces, the luminance is specified by a single number whose range varies from 0 to 65,535. The color black has a luminance value of 0, regardless of the color space.

Colors and Color-Related Objects

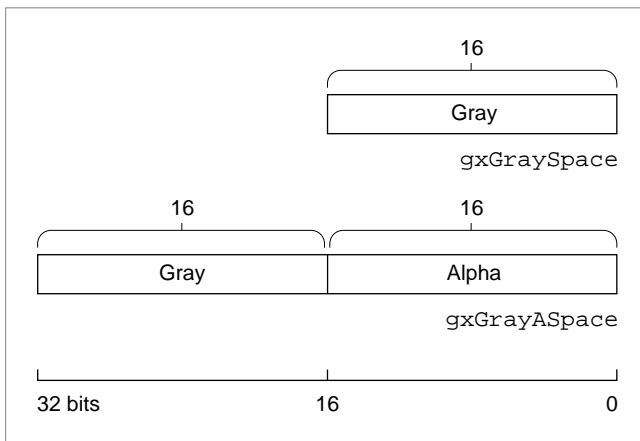
Table 4-1 Luminance-based color spaces supported by QuickDraw GX

Constant	Enumeration Value	Explanation
<code>gxGraySpace</code>	<code>0x000A</code>	16 bits per component (gray only); component values range from 0 to 0xFFFF. Total storage size for each color value: 16 bits.
<code>gxGrayASpace</code>	<code>0x008A</code>	16 bits per component (gray and alpha); component values range from 0 to 0xFFFF. Total storage size for each color value: 32 bits. Alpha channels are described on page 4-24.

Figure 4-2 is a visual representation of the storage formats for the luminance-based color spaces.

Note

This figure and all subsequent storage-format figures in this chapter assume that data storage is “big-endian,” that is, that lower addresses correspond to higher-order bytes in a word or long word value. For processors whose storage model is different, the elements of the figures would be in a different order. These figures are presented for illustrative purposes only, and are not intended to specify details of storage order. ♦

Figure 4-2 Storage formats for luminance-based color spaces

Colors and Color-Related Objects

QuickDraw GX does not support an 8-bit luminance-based color space because such a color space can be more conveniently represented as an indexed color space with a color set. Indexed color space is described in the section “Indexed Color Spaces” beginning on page 4-22; color sets are described in the section “When Color Matching Occurs” beginning on page 4-31.

RGB-Based Color Spaces

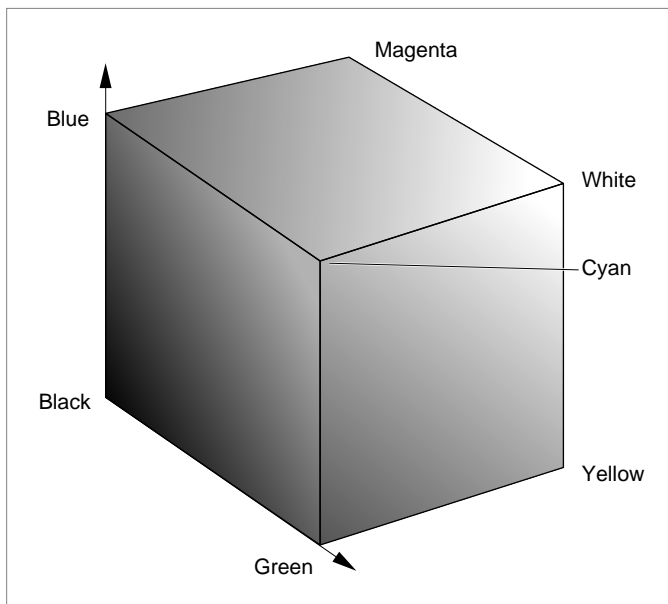
RGB-based color spaces are the most commonly used color spaces in computer graphics, primarily because they are directly supported by most color monitors. The groups of color spaces within the RGB base family include

- RGB spaces
- HSV and HLS spaces

RGB Spaces

Any color expressed in *RGB space* is some mixture of three primary colors red, green, and blue. Most RGB-based color spaces can be visualized as a cube, as in Figure 4-3, with corners of black, the three primaries (red, green, and blue), the three secondaries (cyan, magenta, and yellow), and white. See, for example, Figure 4-3; see also Color Plate 4 at the front of this book.

Figure 4-3 RGB color space

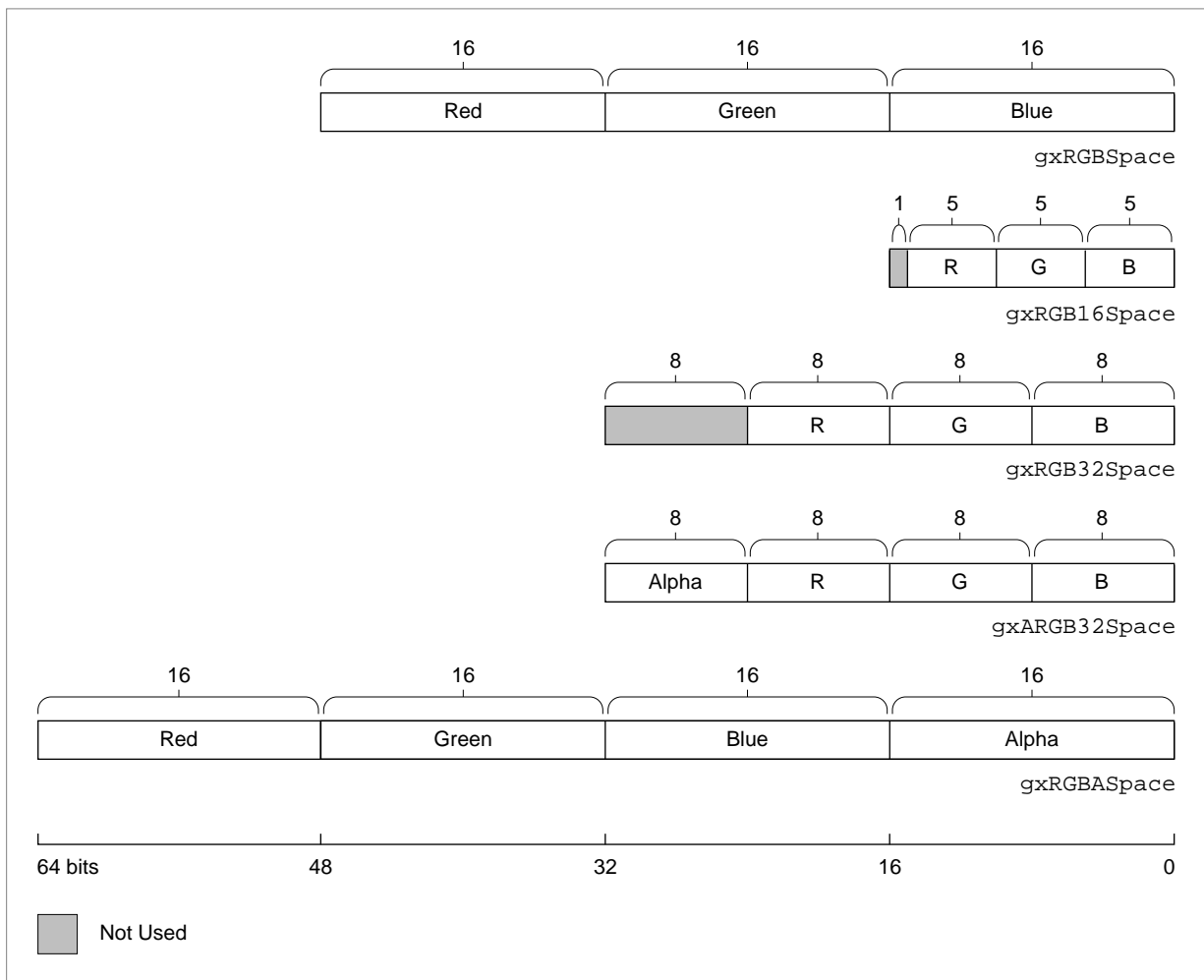


Colors and Color-Related Objects

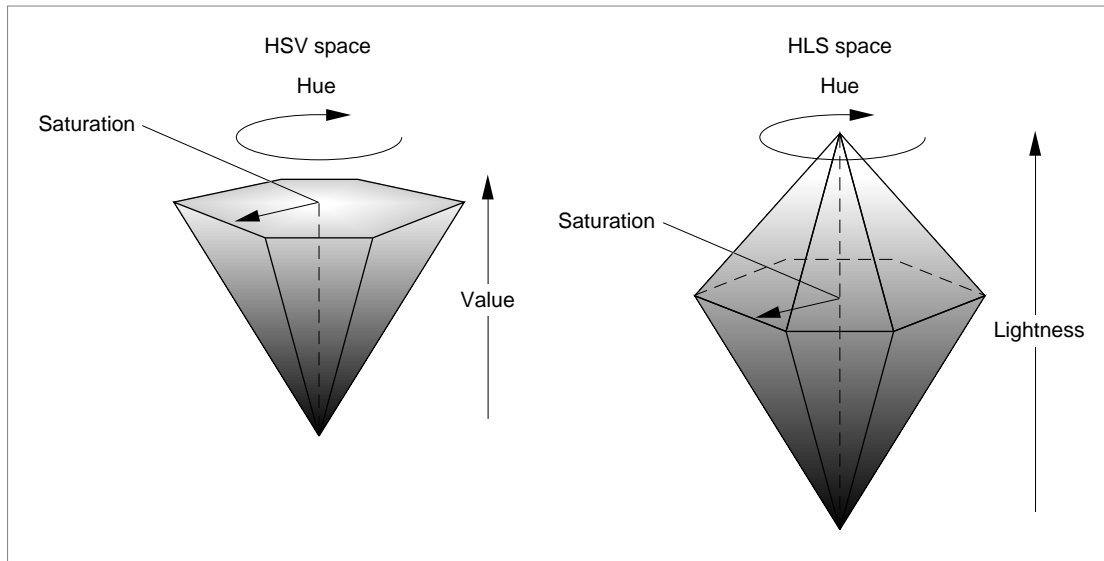
The RGB color spaces supported by QuickDraw GX (and defined in the `gxColorSpaces` enumeration) are `gxRGBSpace`, `gxRGB16Space`, `gxRGB32Space`, `gxRGBASpace`, and `gxARGB32Space`. See Table 4-2 and Figure 4-4 for storage-format details. In each of these spaces, a color value is represented by three or four color components, representing red, green, blue (and in some cases alpha); each component can vary in the number of bits used for its storage. The color black is represented by component values of 0 in the red, green, and blue components.

Table 4-2 RGB color spaces supported by QuickDraw GX

Constant	Enumeration Value	Explanation
<code>gxRGBSpace</code>	0x0001	16 bits per component (red, green, and blue); component values range from 0 to 0xFFFF. Total storage size for each color value: 48 bits.
<code>gxRGB16Space</code>	0x0501	5 bits per component (red, green, and blue); component values range from 0 to 0x1F. Total storage size for each color value: 16 bits (bit 15 is not used).
<code>gxRGB32Space</code>	0x0801	8 bits per component (red, green, and blue); component values range from 0 to 0xFF. Total storage size for each color value: 32 bits (bits 24–31 are not used).
<code>gxARGB32Space</code>	0x1881	8 bits per component (red, green, blue, and alpha); component values range from 0 to 0xFF. Total storage size for each color value: 32 bits. Alpha channels are described on page 4-24.
<code>gxRGBASpace</code>	0x0081	16 bits per component (red, green, blue, and alpha); component values range from 0 to 0xFFFF. Total storage size for each color value: 64 bits. Alpha channels are described on page 4-24.

Figure 4-4 Storage formats for RGB color spaces**HSV and HLS Color Spaces**

HSV space and *HLS space* are transformations of RGB space that allow colors to be described in terms more natural to an artist. The name *HSV* stands for *hue*, *saturation*, and *value*, and *HLS* stands for *hue*, *lightness*, and *saturation*. The two spaces can be thought of as being single and double cones, as shown in Figure 4-5. (See also Color Plate 4 at the front of this book for a slightly different representation of these color spaces.)

Figure 4-5 HSV color space and HLS color space

The components in HLS space are analogous, but not completely identical, to the components in HSV space:

- The hue component in both color spaces is an angular measurement, analogous to position around a color wheel. A hue value of 0 indicates the color red; the color green is at a value corresponding to 120°, and the color blue is at a value corresponding to 240°. Horizontal planes through the cones in Figure 4-5 are hexagons; the primaries and secondaries (red, yellow, green, cyan, blue, and magenta) occur at the vertices of the hexagons.
- The saturation component in both color spaces describes color intensity. A saturation value of 0 (in the middle of a hexagon) means that the color is “colorless” (gray); a saturation value at the maximum (at the outer edge of a hexagon) means that the color is at maximum “colorfulness” for that hue angle and brightness.
- The value component (in HSV space) and the lightness component (in HLS space) describe brightness or luminance. In both color spaces, a value of 0 represents black. In HSV space, a maximum value for value means that the color is at its brightest. In HLS space, a maximum value for lightness means that the color is white, regardless of the current values of the hue and saturation components. The brightest, most intense color in HLS space occurs at a lightness value of exactly half the maximum.

Colors and Color-Related Objects

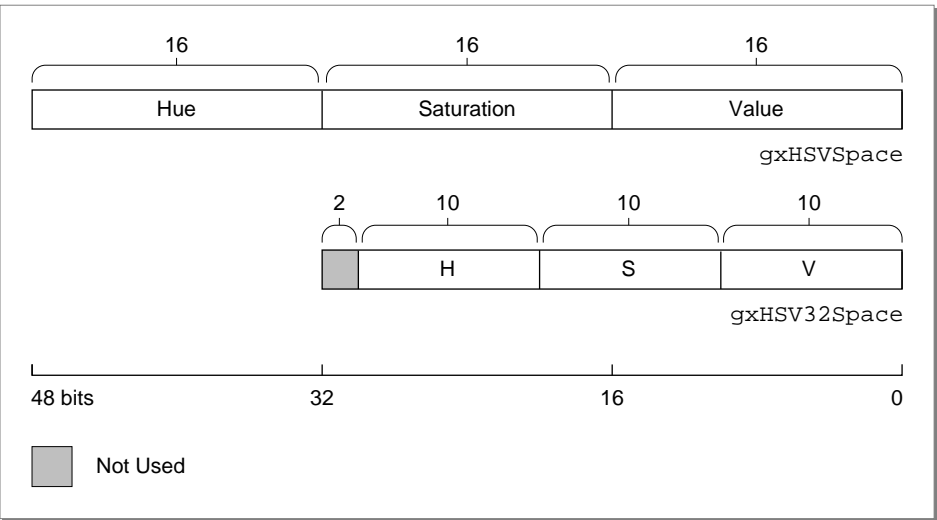
The HLS and HSV color spaces supported by QuickDraw GX (and defined in the `gxColorSpaces` enumeration) are `gxHSVSpace`, `gxHLSSpace`, `gxHSV32Space`, and `gxHLS32Space`. See Table 4-3 and Figure 4-6 for details of storage format.

Table 4-3 HSV and HLS color spaces supported by QuickDraw GX

Constant	Enumeration Value	Explanation
<code>gxHSVSpace</code>	<code>0x0003</code>	16 bits per component (hue, saturation, and value); component values range from 0 to 0xFFFF. Total storage size for each color value: 48 bits.
<code>gxHLSSpace</code>	<code>0x0004</code>	16 bits per component (hue, lightness, and saturation); component values range from 0 to 0xFFFF. Total storage size for each color value: 48 bits.
<code>gxHSV32Space</code>	<code>0x0A03</code>	10 bits per component (hue, saturation, and value); component values range from 0 to 0x3FF. Total storage size for each color value: 32 bits (bits 30–31 are not used).
<code>gxHLS32Space</code>	<code>0x0A04</code>	10 bits per component (hue, lightness, and saturation); component values range from 0 to 0x3FF. Total storage size for each color value: 32 bits (bits 30–31 are not used).

Figure 4-6 shows storage formats for the supported HSV color spaces. Formats for the HLS spaces are identical.

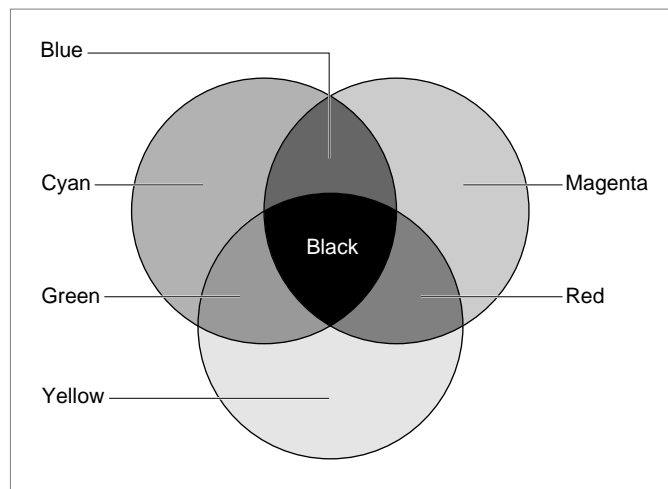
Figure 4-6 Storage formats for HSV color spaces



CMYK Color Spaces

CMYK space is a color space that models the way ink builds up in printing. The name **CMYK** refers to cyan, magenta, yellow, and black. Cyan, magenta, and yellow are the three primary colors in this color space, and red, green, and blue are the three secondaries. Theoretically black is not needed. However, when full-saturation cyan, magenta, and yellow inks are mixed equally on paper, the result is usually a dark brown, rather than black. Therefore, black ink is overprinted in darker areas to give a better appearance. Figure 4-7 shows how the primary colors in CMYK space mix to form other colors. (See also Color Plate 4 at the front of this book.)

Figure 4-7 Colors in CMYK color space



Theoretically, the relation between RGB values and CMY values in CMYK space is quite simple:

```
Cyan      = 1.0 - red;
Magenta   = 1.0 - green;
Yellow    = 1.0 - blue;
```

(where red, green, and blue intensities are expressed as fractional values varying from 0 to 1). In reality, the process of deriving the cyan, magenta, yellow, and black values from a color expressed in RGB space is complex, involving device-specific, ink-specific, and even paper-specific calculations of the amount of black to add in dark areas (**black generation**), and the amount of other ink to remove (**undercolor removal**) where black is to be printed. QuickDraw GX performs those calculations for you when converting among color spaces, commonly using color profiles as described in the section “Color Profiles” beginning on page 4-28.

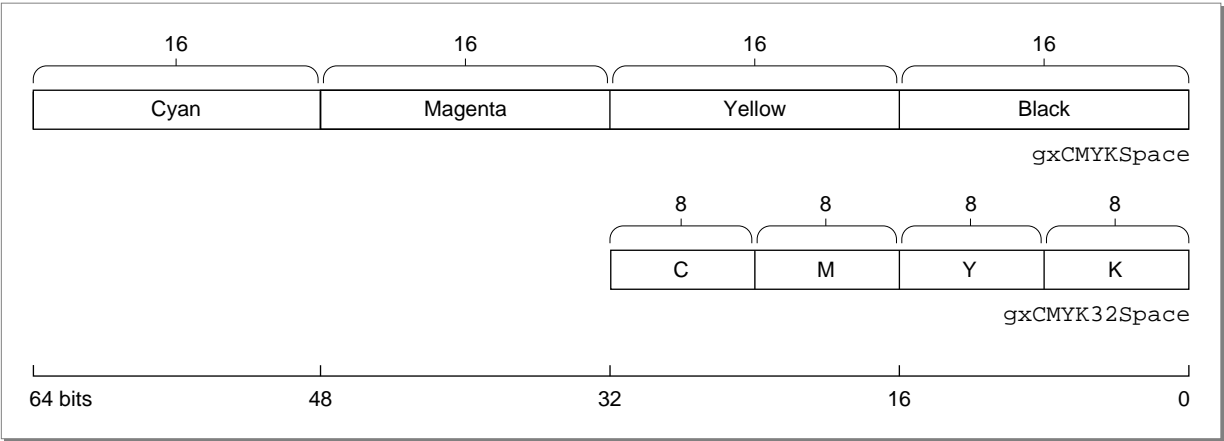
Colors and Color-Related Objects

The CMYK color spaces supported by QuickDraw GX (and defined in the `gxColorSpaces` enumeration) are `gxCMYKSpace` and `gxCMYK32Space`. See Table 4-4 and Figure 4-8 for details of storage format.

Table 4-4 CMYK color spaces supported by QuickDraw GX

Constant	Enumeration Value	Explanation
<code>gxCMYKSpace</code>	<code>0x0002</code>	16 bits per component (cyan, magenta, yellow, and black); component values range from 0 to 0xFFFF. Total storage size for each color value: 64 bits.
<code>gxCMYK32Space</code>	<code>0x0802</code>	8 bits per component (cyan, magenta, yellow, and black); component values range from 0 to 0xFF. Total storage size for each color value: 64 bits.

Figure 4-8 Storage formats for CMYK color spaces



Universal Color Spaces

Some color spaces allow color to be expressed in a device-independent way. Whereas RGB colors vary with monitor characteristics, and CMYK colors vary with printer and paper characteristics, *universal colors* are meant to be true representations of colors as perceived by the human eye. These color representations, called *universal color spaces*, result from work carried out in 1931 by the Commission Internationale d’Eclairage (CIE), and for that reason are also called *CIE-based color spaces*.

In addition, broadcast-video color space (YIQ) is based on device-independent color characteristics, in that its colors are measured in terms of a standard device. It is therefore considered universal and is discussed in this section.

Colors and Color-Related Objects

XYZ Space

There are several CIE-based color spaces, but all are derived from the fundamental **XYZ space**. The XYZ space allows colors to be expressed as a mixture of the three *tristimulus values* X, Y, and Z. The term *tristimulus* comes from the fact that color perception results from the retina of the eye responding to three types of stimuli. After experimentation, the CIE set up a hypothetical set of primaries, XYZ, that correspond to the way the eye's retina behaves.

The CIE defined the primaries so that all visible light maps into a positive mixture of X, Y, and Z, and so that Y correlates approximately to the apparent lightness of a color. Generally, the mixtures of X, Y, and Z components used to describe a color are expressed as percentages ranging from 0% up to, in some cases, just over 100%.

Other universal color spaces based on XYZ space are used primarily to relate some particular aspect of color or some perceptual color difference to XYZ values.

Yxy Space

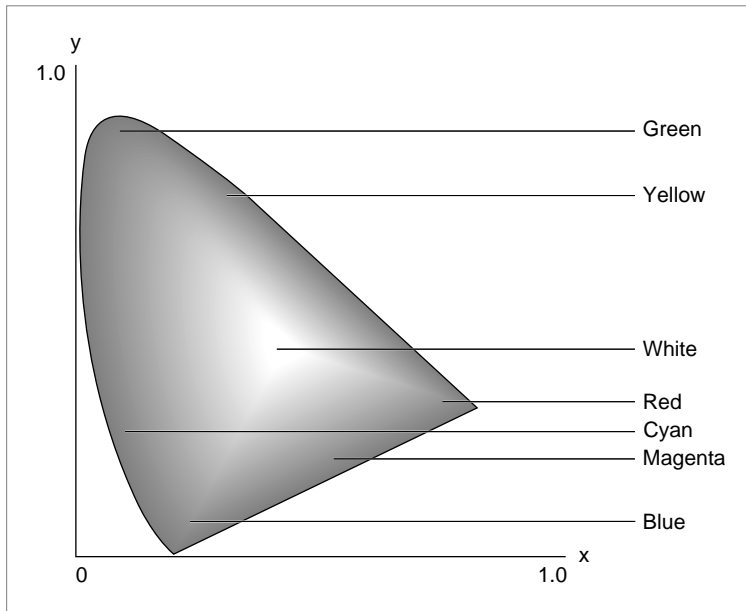
Yxy space expresses the XYZ values in terms of x and y *chromaticity* coordinates, somewhat analogous to the hue and saturation coordinates of HSV space. The coordinates are shown in the following formulas, used to convert XYZ into Yxy:

$$Y = Y$$

$$x = X / (X+Y+Z)$$

$$y = Y / (X+Y+Z)$$

Note that the Z tristimulus value is incorporated into the new coordinates, and does not appear by itself. Since Y still correlates to the lightness of a color, the other aspects of the color are found in the chromaticity coordinates x and y. This allows color variation in Yxy space to be plotted on a two-dimensional diagram. Figure 4-9 shows the layout of colors in the x and y plane of Yxy space. Color Plate 4 at the front of this book shows the same plot in color.

Figure 4-9 Yxy chromaticities

L*u*v* Space and L*a*b* Space

One problem with representing colors using the XYZ and Yxy color spaces is that they are perceptually nonlinear: it is not possible to accurately evaluate the perceptual closeness of colors based on their relative positions in XYZ or Yxy space. Colors that are close together in Yxy space may seem very different to observers, and colors that seem very similar to observers may be widely separated in Yxy space.

*L*u*v* space* is a nonlinear transformation of XYZ space in order to create a perceptually linear color space. *L*a*b* space* is a nonlinear transformation (a third-order approximation) of the Munsell color-notation system (not described here). Both are designed to match perceived color difference with quantitative distance in color space.

Both *L*u*v* space* and *L*a*b* space* represent colors relative to a *reference white point*, which is a specific definition of what is considered white light, represented in terms of XYZ space, and usually based on the whitest light that can be generated by a given device. (In that sense *L*u*v** and *L*a*b** are not completely device independent; two numerically equal colors are truly identical only if they were measured relative to the same white point.)

Measuring colors in relation to a white point allows for color measurement under a variety of illuminations. The luminance of the white point of the QuickDraw GX default color profile matches the luminance of the white point on the Apple 13-inch color monitor. Color profiles are described in the section “Color Conversion and Color Matching” beginning on page 4-26.

Colors and Color-Related Objects

A primary benefit of using $L^*u^*v^*$ space and $L^*a^*b^*$ space is that the perceived difference between any two colors is proportional to the geometric distance in the color space between their color values. For applications where closeness of color needs to be quantified, such as in colorimetry, gemstone evaluation, or dye matching, use of $L^*u^*v^*$ space or $L^*a^*b^*$ space is common.

The formulas for transforming an XYZ color into an $L^*u^*v^*$ color are

```
if (Y/Yn > 0.008856)
    L = 116.0 * (Y / Yn)1/3 - 16.0;
else
    L = 903.3 * (Y / Yn);
u = 13.0 * L * (u' - u'_n);
v = 13.0 * L * (v' - v'_n);
```

where

```
u' = 4 * x / (X + 15*Y + 3*Z);
v' = 9 * y / (X + 15*Y + 3*Z);
```

and u'_n , v'_n , and Y_n are the u' , v' , and Y values for the reference white point.

Similarly, the formulas for transforming an XYZ color into an $L^*a^*b^*$ color are

```
if (Y/Yn > 0.008856)
    L = 116.0 * (Y / Yn)1/3 - 16.0;
else
    L = 903.3 * (Y / Yn);
a = 500.0 * ( (X / Xn)1/3 - (Y / Yn)1/3 );
b = 500.0 * ( (Y / Yn)1/3 - (Z / Zn)1/3 );
```

where X_n , Y_n , and Z_n are the XYZ values for the reference white point.

Formats for XYZ-Based Color Spaces

The universal color spaces supported by QuickDraw GX (and defined in the `gxColorSpaces` enumeration) are `gxYXYSpace`, `gxXYZSpace`, `gxLUVSpace`, `gxLABSpace`, `gxYXY32Space`, `gxXYZ32Space`, `gxLUV32Space`, and `gxLAB32Space`. See Table 4-5 and Figure 4-10 for details of storage format. Note that the ranges of values for the components differ significantly among the different color spaces.

Figure 4-10 shows storage formats for the supported XYZ color spaces. Formats for the Yxy , $L^*u^*v^*$, and $L^*a^*b^*$ spaces are identical.

Table 4-5 Universal color spaces supported by QuickDraw GX

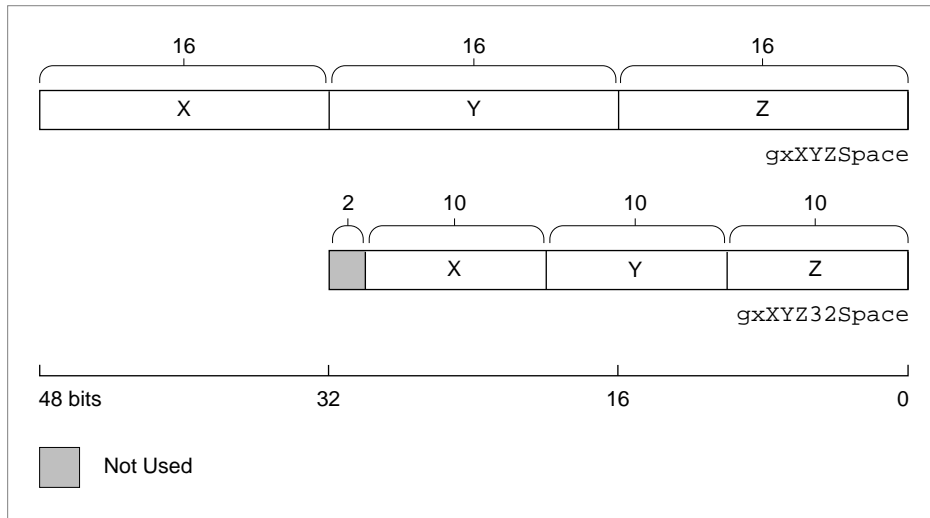
Constant	Enumeration Value	Explanation
gxYXYSpace	0x0005	16 bits per component (Y, x, and y); component values range from 0 (0%) to 0xFFFF (100%). Total storage size for each color value: 48 bits.
gxXYZSpace	0x0006	16 bits per component (X, Y, and Z). Component values range from 0 (0%) to 0xFFFF (200%; a value of 0x8000 represents 100%). Total storage size for each color value: 48 bits.
gxLUVSpace	0x0007	16 bits per component (L*, u*, and v*). The L* component values range from 0 (0%) to 0xFFFF (100% of white-point luminance). The u* and v* component values range from 0 (–1) to 0xFFFF (+1). Total storage size for each color value: 48 bits.
gxLABSpace	0x0008	16 bits per component (L*, a*, and b*). The L* component values range from 0 (0%) to 0xFFFF (100% of white-point luminance). The a* and b* component values range from 0 (–1) to 0xFFFF (+1). Total storage size for each color value: 48 bits.
gxYXY32Space	0x0A05	10 bits per component (Y, x, and y); component values range from 0 (0%) to 0x3FF (100%). Total storage size for each color value: 32 bits (bits 30 and 31 not used).
gxXYZ32Space	0x0A06	10 bits per component (X, Y, and Z). Component values range from 0 (0%) to 0x3FF (200%; a value of 0x200 represents 100%). Total storage size for each color value: 32 bits (bits 30 and 31 not used).
gxLUV32Space	0x0A07	10 bits per component (L*, u*, and v*). The L* component values range from 0 (0%) to 0x3FF (100% of white-point luminance). The u* and v* component values range from 0 (–1) to 0x3FF (+1). Total storage size for each color value: 32 bits (bits 30 and 31 not used).
gxLAB32Space	0x0A08	10 bits per component (L*, a*, and b*). The L* component values range from 0 (0%) to 0x3FF (100% of white-point luminance). The a* and b* component values range from 0 (–1) to 0x3FF (+1). Total storage size for each color value: 32 bits (bits 30 and 31 not used).

NOTE Because u*, v*, a*, and b* are normally signed numbers between 1.0 and -1.0, you can convert them into shorts as follows:

```
anUnsignedshort = ((aFloat + 1.0)/2) * 65535.0;
```

Figure 4-10 Storage formats for XYZ color spaces

Colors and Color-Related Objects

**Video Color Spaces**

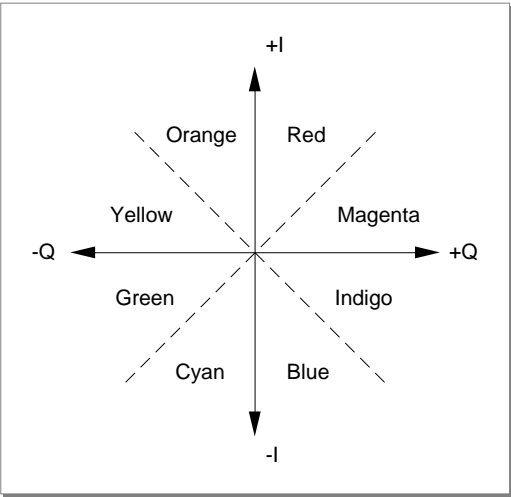
YIQ space is sometimes called *video color space*. It is based on the way a specific kind of RGB data is broken down for color television transmission. The three dimensions that describe these color spaces are Y, I, and Q, in which Y represents luminance and the other two components carry color information.

Because the Y channel represents luminance it can be used alone; the Y channel is the only channel used in black and white television. The I and Q channels are called *color difference* channels: the Y channel is split between them. The notations “I” and “Q” stand for “in phase” and “in quadrature,” respectively, referring to the method by which all of the channels are combined into a signal for broadcast.

QuickDraw GX also defines NTSC and PAL color spaces. NTSC space corresponds to the color encoding used for color broadcasting in the United States, whereas PAL space corresponds to the color encoding used in Europe. NTSC and PAL have different screen resolutions, frequencies, and are otherwise incompatible, but in terms of how color values are calculated, NTSC space and PAL space are both identical to YIQ space.

In YIQ space, the Y component can vary from 0 (black) to its maximum value (full luminance). I and Q are normally signed values, so they are centered around 0. Figure 4-11 illustrates how colors map into the I and Q dimensions of YIQ space.

Figure 4-11 The I and Q axes in YIQ color space



The video color spaces supported by QuickDraw GX (and defined in the `gxColorSpaces` enumeration) are `gxYIQSpace`, `gxNTSCSpace`, `gxPALSpace`, `gxYIQ32Space`, `gxNTSC32Space`, and `gxPAL32Space`. See Table 4-6 and Figure 4-12 for details of storage format. In each of these spaces, a color value is represented by Y, I, and Q color components.

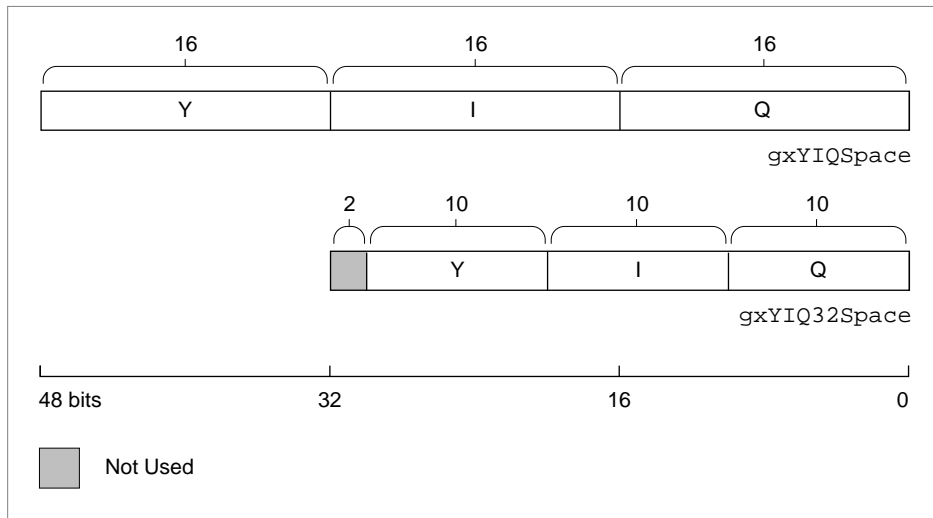
Table 4-6 Video color spaces supported by QuickDraw GX

Constant	Enumeration Value	Explanation
<code>gxYIQSpace</code>	<code>0x0009</code>	16 bits per component (Y, I, and Q); Y-component values range from 0 to <code>0xFFFF</code> ; I- and Q-component values range from <code>-0x7FFF</code> to <code>+0x7FFF</code> . Total storage size for each color value: 48 bits.
<code>gxNTSCSpace</code>	<code>0x0009</code>	(same as <code>gxYIQSpace</code>)
<code>gxPALSpace</code>	<code>0x0009</code>	(same as <code>gxYIQSpace</code>)
<code>gxYIQ32Space</code>	<code>0x0A09</code>	10 bits per component (Y, I, and Q); Y-component values range from 0 to <code>0x3FF</code> ; I- and Q-component values range from <code>-0x1FF</code> to <code>+0x1FF</code> . Total storage size for each color value: 32 bits (bits 30 and 31 are not used).
<code>gxNTSC32Space</code>	<code>0x0A09</code>	(same as <code>gxYIQ32Space</code>)
<code>gxPAL32Space</code>	<code>0x0A09</code>	(same as <code>gxYIQ32Space</code>)

Colors and Color-Related Objects

Figure 4-12 shows storage formats for the supported YIQ color spaces. Formats for the NTSC and PAL spaces are identical.

Figure 4-12 Storage formats for YIQ color spaces



You can find more information on the theories of color and the various color spaces in the following publications:

Measuring Color, by R.W.G. Hunt, John Wiley & Sons, New York, 1987.

Illumination and Color in Computer Generated Imagery, by Roy Hall, Springer-Verlag, New York, 1989.

Indexed Color Spaces

In situations where you use only a limited number of colors, it can be impractical or impossible to specify colors directly. For example, if you have a bitmap with only a few bits per pixel (1, 2, 4 or 8 for QuickDraw GX), each pixel is too small to contain a complete color specification; its color must be specified as an index into a list or table of color values. If you are using spot colors in printing or pen colors in plotting, it can be simpler and more precise to specify each color as an index into a list instead of an actual color value. Also, if you want to restrict the user to drawing with a specific set of colors, you can put them in a list and specify them by index.

Indexed space is the color space you use when drawing with indirectly specified colors. An indexed color value (a color specification in indexed color space) consists of an index value and a reference to a color set object. The color set contains a list of color values and a specification of the color space for those color values; the index value specifies which color to use from the list. Color values are defined in the section “Color-Component Values, Color Values, and Colors” beginning on page 4-25. Color set objects are described in the section “About Color Set Objects” beginning on page 4-32.

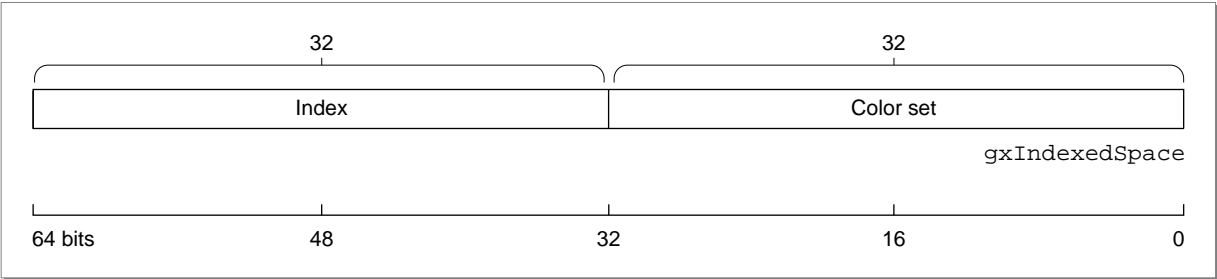
Colors and Color-Related Objects

QuickDraw GX supports the single indexed color space format `gxIndexedSpace` (defined in the `gxColorSpaces` enumeration). See Table 4-7 and Figure 4-4 for details of storage format. Although there is a single format for indexed color space, you can create any number of unique indexed color spaces, using different sets of colors from any of the defined color spaces.

Table 4-7 Indexed color space supported by QuickDraw GX

Constant	Enumeration Value	Explanation
<code>gxIndexedSpace</code>	<code>0x000B</code>	Indicates that the color value is a (1-based) index into the referenced color set. Total storage size for each color value: 64 bits.

Figure 4-13 Storage format for indexed color space



Color spaces and bitmaps

Bitmaps commonly use indexed color space, but if pixel size is large enough a bitmap can specify colors directly in any color space. These are the restrictions on the use of color spaces with bitmaps:

- Bitmaps with 1, 2, 4, or 8 bits per pixel must use `gxIndexedSpace`.
- Bitmaps with 16 bits per pixel can use `gxRGB16Space`. They cannot use `gxIndexedSpace`.
- Bitmaps with 32 bits per pixel can use `gxRGB32Space`, `gxARGB32Space`, `gxCMYK32Space`, `gxHSV32Space`, `gxHLS32Space`, `gxYXY32Space`, `gxXYZ32Space`, `gxLUV32Space`, `gxLAB32Space`, `gxYIQ32Space`, `gxNTSC32Space`, or `gxPAL32Space`—that is, all defined 32-bit color spaces. They cannot use `gxIndexedSpace`.
- Hardware devices that have 24 bits of physical memory per pixel can support `gxRGB32Space`. Hardware devices that have 32 bits of physical memory per pixel can support `gxRGB32Space` plus all the other defined 32-bit color spaces.

Bitmaps are described further in the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. ♦

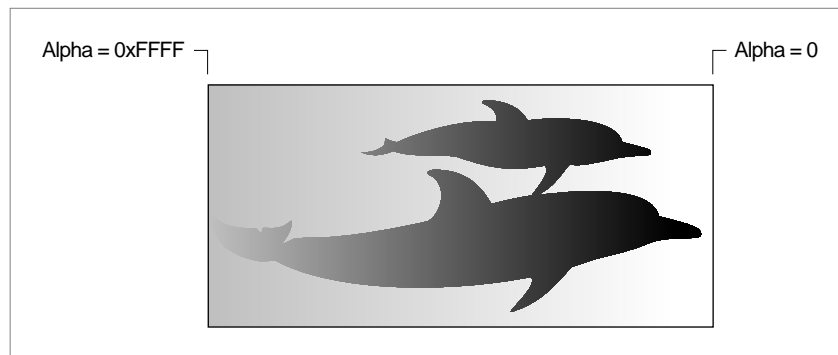
Color Spaces With Alpha Channels

QuickDraw GX supports the use of an alpha channel in one luminance-based color space (`gxGrayASpace`) and two RGB color spaces (`gxRGBASpace` and `gxARGB32Space`). An *alpha channel* is a component in a color space whose value typically determines the opacity of the color expressed by the other components. An alpha-channel value of 0 in a color means that the color is completely transparent, and a maximum value means that the color is completely opaque. A value in between means that the color is partially transparent.

How transparency is handled in drawing depends on the transfer mode used when the color is drawn. (Transfer modes are discussed in the chapter “Ink Objects” in this book.) Typically, however, transparency in a color being drawn—the source color—means that the existing color at the location where drawing occurs—the destination color—shows through. Where the source is completely opaque, the destination is completely covered and is invisible; where the source is completely transparent, the destination shows through unchanged and the source is invisible.

Figure 4-14 shows an example in which a uniform gray image (in `gxGrayASpace`) is drawn over a black-and-white image. The gray color of the source is uniform across the rectangle, but the alpha-channel value decreases from 0xFFFF on the left to 0 on the right. As the alpha value decreases rightward, more and more of the destination color shows through. (Color Plate 2 at the front of this book shows a similar drawing example in color.)

Figure 4-14 Showing color transparency with an alpha channel



For more information on using alpha channels to achieve particular drawing effects, see the chapter “Ink Objects” in this book.

Color-Component Values, Color Values, and Colors

Each of the color spaces described in this chapter requires one or more numeric values in a particular format to specify a color. This section describes the data types and structures with which QuickDraw GX describes colors in its color spaces.

Each dimension, or component, in a color space has a *color-component value*. In the fundamental, unpacked QuickDraw GX color spaces—those with 16 bits per component—each color-component value is of type `gxColorValue`:

```
typedef unsigned short gxColorValue;
```

A color-component value can vary from 0 to 65,535 (0xFFFF), although the numerical interpretation of that range is different for different color spaces, as has been noted in Table 4-1 through Table 4-7. In most cases, color-component intensities are interpreted numerically as varying between 0 and 1.0; for that reason, QuickDraw GX provides the constant `gxColorValue1` to represent 0xFFFF.

Depending on the color space used, one, two, three, or four color-component values combine to make a *color value*. A color value is a structure; it is the complete specification of a color in a given color space. QuickDraw GX supports 13 color-value formats, representing the fundamental 16-bits-per-component color spaces; all color operations in memory use one of those formats. The color-value formats are described in the section “Color Values” beginning on page 4-50. For example, an RGB color value has this format:

```
struct gxRGBColor{
    gxColorValue    red;
    gxColorValue    green;
    gxColorValue    blue;
};
```

This is exactly the storage format for colors in `gxRGBSpace`. However, colors stored in `gxRGB16Space` or `gxRGB32Space` have a packed storage format, and need to be converted to `gxRGBColor` format when they are used. QuickDraw GX takes care of this for you; as far as your application is concerned, you can always manipulate colors in the color space you have specified.

Colors and Color-Related Objects

A color value plus a specification of the color space it belongs to (plus an optional reference to a color profile to use for color matching) constitute a *color* in QuickDraw GX. A color is defined by the `gxColor` structure:

```
struct gxColor{
    gxColorSpace      space;
    gxColorProfile    profile;
    union {
        struct gxCMYKColor    cmyk;
        struct gxRGBColor     rgb;
        struct gxRGBAColor    rgba;
        struct gxHSVColor     hsv;
        struct gxHLSColor     hls;
        struct gxXYZColor     xyz;
        struct gxYXYColor     yxy;
        struct gxLUVColor     luv;
        struct gxLABColor     lab;
        struct gxYIQColor     yiq;
        gxColorValue          gray;
        struct gxGrayAColor    graya;
        unsigned short        pixel16;
        unsigned long         pixel32;
        struct gxIndexedColor  indexed;
        gxColorValue          component[4];
    } element;
};
```

Each `gxColor` structure holds the specification of a single color. Note that, besides the basic color-value formats such as `gxRGBColor` and `gxXYZColor`, a QuickDraw GX color can contain a 16-bit or 32-bit pixel value or an indexed color value, and you can also access the color as an array of color-component values. Each of the color values in the `element` union of the `gxColor` structure is described in the section “The Color Structure” beginning on page 4-53.

Color Conversion and Color Matching

Color support in QuickDraw GX is designed for device independence. You can work in whatever color space is most convenient for you, you can convert colors from one color space to another, and you can input and output colors with a variety of physical devices with minimum error and loss of information.

Colors and Color-Related Objects

You may want to explicitly convert from one color space to another for a variety of reasons, such as

- to allow users to work in a more familiar context (perhaps HSV instead of RGB)
- to convert device-dependent colors to device-independent colors (such as RGB to $L^*u^*v^*$)
- to preview printed output onscreen (by converting RGB to CMYK)
- to display on monochrome monitors or printers (by converting to gray space)

In addition, QuickDraw GX automatically converts colors from one space to another whenever necessary, such as when it displays a color that is defined in terms of one space on a device whose colors are defined in terms of another space.

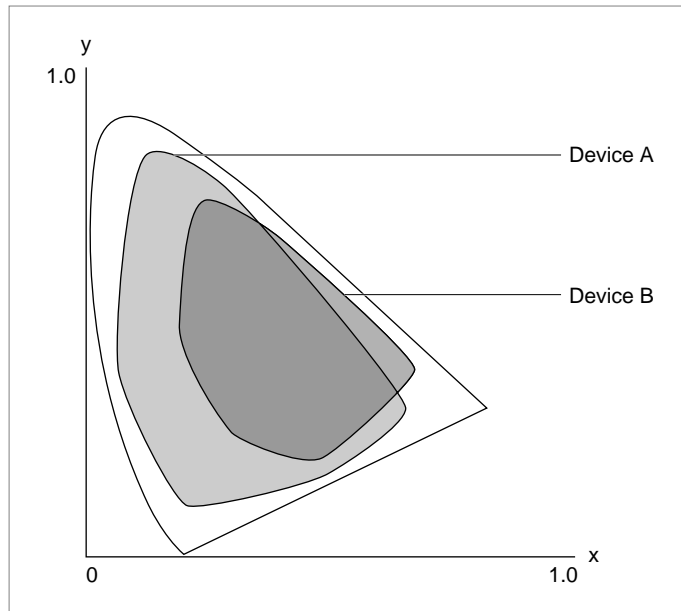
When converting among color spaces within a base family (such as HSV to RGB) and for display on the same device, the conversion is exact and there is no loss or error.

However, when converting across base families (such as RGB to CMYK, or HLS to XYZ), and when converting within the same family but across different display devices, device dependence is introduced and must be accounted for.

Different imaging devices (scanners, monitors, printers) work in different color spaces and each can have a different *gamut*, or range of colors that it can produce. Monitors from different manufacturers all display colors in RGB, but may have different RGB gamuts. Printers that work in CMYK space vary drastically in their gamuts, especially if they use different printing technologies. Even a single printer's gamut can vary significantly with the ink or type of paper it uses. It's easy to see that conversion from RGB colors on an individual monitor to CMYK colors on an individual printer using a particular paper type can lead to unpredictable results.

When an image is output to a particular device, the device displays only those colors that are within its gamut. Likewise, when an image is created by scanning, only those colors within the scanner's gamut are saved. Devices with different gamuts cannot reproduce each others' colors exactly, but careful shifting of the colors used on one device can improve the visual match when the image is displayed on another.

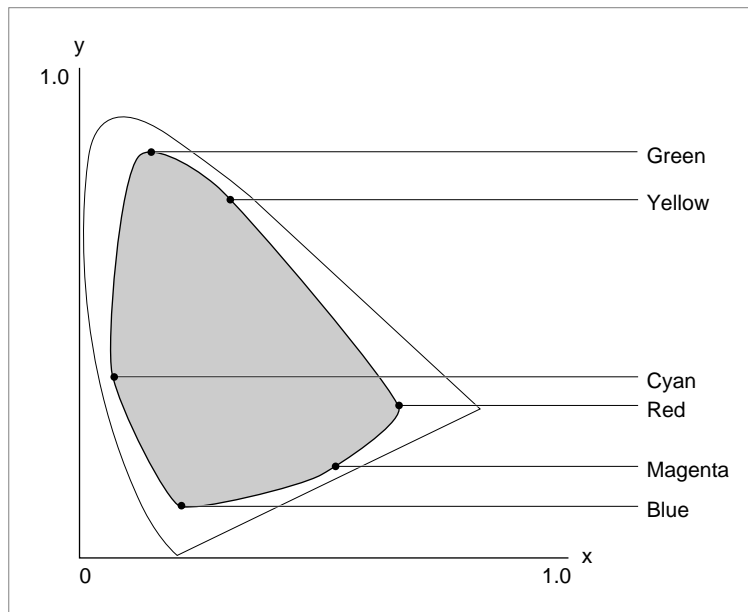
Figure 4-15 shows examples of two devices' color gamuts, projected onto Yxy space. Both devices produce less than the total possible range of colors, and device B is restricted to a significantly smaller range than device A. The problem illustrated by Figure 4-15 is to be able to display the same image on both devices with a minimum of visual mismatch. The solution to the problem is the use of color profiles and color-matching methods.

Figure 4-15 Color gamuts for two devices (in Yxy space)

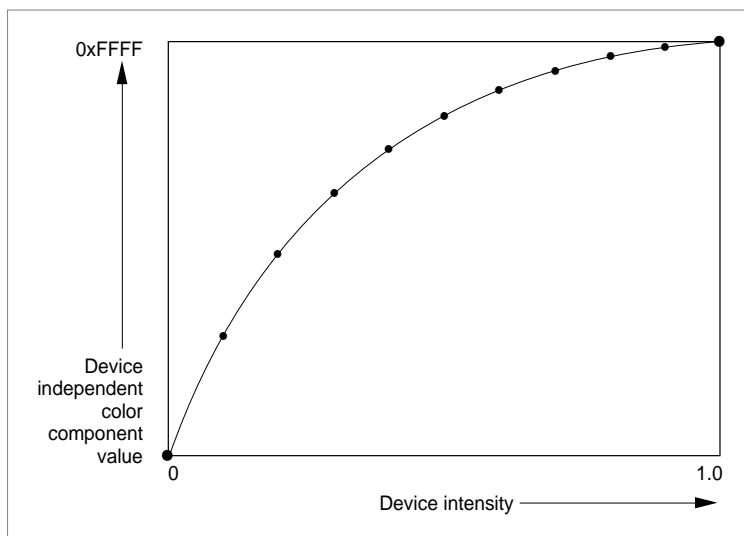
Color Profiles

Converting colors accurately across different input or display devices is called **color matching**. To perform color matching requires the use of a color profile for each device involved. A **color profile** describes the characteristics of a color space for a particular physical device in a particular state. A monitor, for example, might have a single color profile, whereas a printer might have a different profile for each paper type or ink type it uses. A **color-matching method** uses a color profile to convert a color in a given color space on a given device to or from another color space or device, perhaps a device-independent color space.

Different color profiles can have different kinds of information in them. However, any color profile has at least two parts: a set of profile chromaticities and a set of profile response curves. The **profile chromaticities** are color values that define the extremes of saturation that the device can produce for its primary and secondary colors (red, green, blue, cyan, magenta, yellow). Each color value is typically described in terms of a device-independent space such as XYZ. You can think of the profile's chromaticities as defining points at the extremes of that device's gamut, as shown in Figure 4-16. (The points in Figure 4-16 correspond to the limits of the gamut for device A in Figure 4-15.)

Figure 4-16 Profile chromaticities for a device (in Yxy space)

The *profile response curves* are graphs that describe how the profile chromaticities ramp from no intensity to full intensity (there are additional response curves for undercolor removal and black generation in CMYK space). The response curves are analogous to gamma curves for monitors or dot-pitch curves for printing. Figure 4-17 shows an example of a single response curve.

Figure 4-17 A profile response curve for a device

Colors and Color-Related Objects

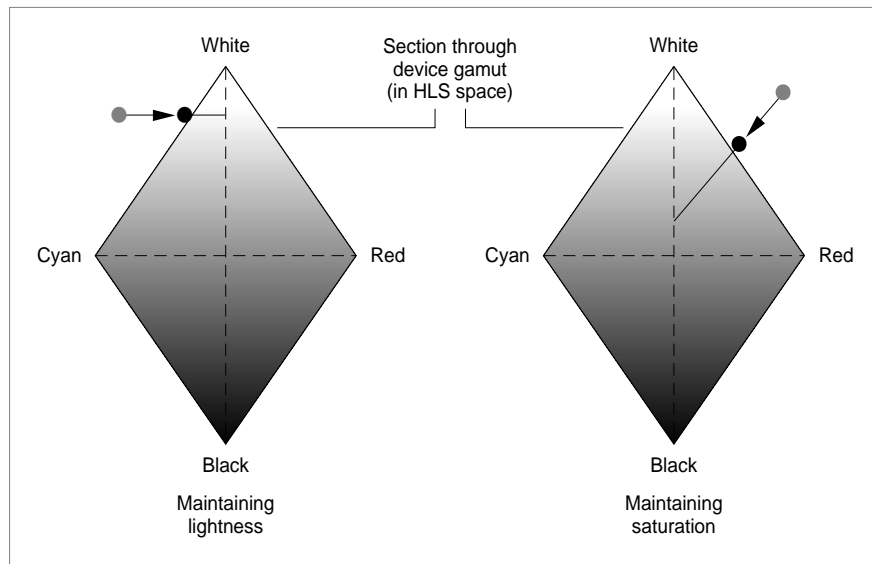
Color profiles contain additional information, such as a specification of how to apply the chromaticities and response curves for matching (see the next section, “Color-Matching Methods”), and a name string. They may also have custom information used by particular color-matching methods. QuickDraw GX uses color profiles following the format defined by the ColorSync Utilities. See the ColorSync Utilities chapter of *Inside Macintosh: Advanced Color Imaging* for more information.

Color profiles are optional; a given color structure may or may not contain a valid reference to a color profile. If a color profile reference is attached, QuickDraw GX uses it when converting or matching colors; if there is no attached profile, QuickDraw GX uses the default QuickDraw GX color profile; see “The Default Color Profile” on page 4-37.

Color-Matching Methods

When colors consistent with one device’s gamut are displayed on a device with a different gamut, as in Figure 4-15 on page 4-28, a color-matching method attempts to minimize the perceived differences in the displayed colors between the two devices. The default Apple color-matching method, as used with the ColorSync Utilities, uses these three approaches:

- **Colorimetric matching.** In this method, colors that fall within the gamuts of both devices are left unchanged. For example, to match an image from device A onto device B in Figure 4-15, only the colors in the gamut of A that fall outside the gamut of B are altered. Colorimetric matching allows some colors in both images to be exactly the same, which is useful when colors must match quantitatively. A disadvantage of colorimetric matching is that many colors may map to a single color. All colors outside the gamut of B in Figure 4-15, for example, would be converted to colors at the edge of its gamut, reducing the total number of colors in the image and possibly greatly altering its appearance. In colorimetric matching, colors outside the gamut are usually converted to colors with the same lightness, but different saturation, at the edge of the gamut. The left side of Figure 4-18 shows how colors are projected in colorimetric matching.
- **Perceptual matching.** In this method, all the colors of a given gamut are shifted to fit within another gamut. The colors maintain their relative positions, so the relationship between colors is maintained. With realistic images such as scanned photographs, perceptual matching produces better results than colorimetric matching in most cases; in Figure 4-15, for example, the eye could compensate for the difference in gamuts between A and B, and a perceptually matched image on B would look very similar to the original image on A. A disadvantage of perceptual matching is that none of the original colors is unchanged in the copy.
- **Saturation matching.** In some computer graphics, such as bar graphs and pie charts, the actual color displayed is less important than its vividness. In this method, the relative saturation of colors is maintained from gamut to gamut. Colors outside the gamut are usually converted to colors with the same saturation, but different lightness, at the edge of the gamut. The right side of Figure 4-18 shows how colors are projected in saturation matching.

Figure 4-18 Maintaining lightness and maintaining saturation in color matching

QuickDraw GX uses the Macintosh ColorSync Utilities for color matching. ColorSync color-matching methods are Component Manager components and support all three kinds of color-matching, and may support other kinds as well. QuickDraw GX color profile objects contain ColorSync color profile structures, and each structure specifies the kind of matching that should be performed with it.

For more information on ColorSync and color-matching methods, see the ColorSync Utilities chapter of *Inside Macintosh: Advanced Color Imaging*. For more information on Component Manager components, see the Component Manager chapter of *Inside Macintosh: More Macintosh Toolbox*.

When Color Matching Occurs

Color profiles are associated with devices. For example, when a QuickDraw GX-aware scanning application creates a scanned image, it produces a bitmap and attaches a color profile object (containing profile information obtained from the scanner driver) to the bitmap. The color profile that is associated with a shape and describes the characteristics of the device on which the shape was created is called the *source profile*. If the colors in the bitmap are subsequently converted to another color space by the scanning application or by another QuickDraw GX application, QuickDraw GX uses that source profile to match the colors when converting. Bitmaps are described in the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

Colors and Color-Related Objects

To display the bitmap requires using another color profile, which is attached to the view device object associated with the output device. (View device objects are described in the chapter “View-Related Objects” in this book.) That color profile is called the *destination profile*. If the bitmap is displayed on a monitor, QuickDraw GX uses the monitor’s color profile, along with the bitmap’s source profile, to match the bitmap’s colors to the monitor’s gamut. If the bitmap is printed, QuickDraw GX uses the printer’s profile to match the bitmap’s colors to the printer, including generating black and removing undercolors where appropriate.

QuickDraw GX color matching occurs automatically, whenever drawing takes place or whenever colors are converted from a color space in one base family to a color space in a different base family. Most applications need not know what profiles, if any, are attached to the colors they manipulate and draw. However, applications can explicitly use color profiles for purposes such as print previewing, or they can allow the user to create custom, modified profiles for special purposes on particular devices. In addition, specialized applications can calibrate display devices and produce color profiles whose information is stored in the devices’ drivers for use by QuickDraw GX. Such applications make use of the ColorSync Utilities to create their profiles.

Color matching is off by default

Color matching can slow drawing speed. For that reason, when you create a view port, the view port attribute `gxEnableMatchPort` is cleared by default. If you want matching to occur when you draw to the screen, you must first set `gxEnableMatchPort`. (Matching occurs when appropriate during printing, regardless of the state of the `gxEnableMatchPort` attribute.) View port attributes are discussed in the chapter “View-Related Devices” in this book. ♦

For more information on color profiles, see the section “About Color Profile Objects” beginning on page 4-35.

About Color Set Objects

A color set is a QuickDraw GX object that contains a list of colors. Color sets exist to provide the colors for indexed color space. Bitmaps and other shapes that use indexed color space specify colors as indexes into a color set. Color sets are the QuickDraw GX equivalents to color tables in other graphics systems.

QuickDraw GX identifies an individual color set object through a color set *reference*. To obtain information about a color set object, you must send its reference as a parameter to a QuickDraw GX function (except that you can determine if two references identify the same color set object simply by comparing them for equality, and you can examine a reference to see if it is `nil`).

Colors and Color-Related Objects

Any QuickDraw GX color (`gxColor` structure) that contains an indexed color value includes a reference to a color set object. If a shape's ink object has a color in indexed color space, the color includes a reference to a color set object. If the bitmap in a view device object uses indexed color values for its pixels, the bitmap includes a reference to a color set object. (View devices are described in the chapter "View-Related Objects" in this book.)

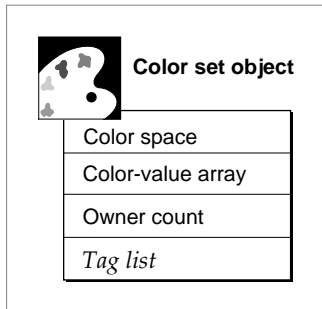
Color sets can be device independent because their colors, like any QuickDraw GX colors, can be matched across devices. The color information is valid for any display device on which the shapes the color sets apply to are drawn.

Color Set Properties

The interface to color set objects is entirely procedural. You manipulate the information in a color set by modifying its properties using QuickDraw GX functions.

Color set objects have four accessible properties, as shown in Figure 4-19. Note that, because a color set is an object and not a data structure, the order of the properties as shown in Figure 4-19 is completely arbitrary. Properties in italics are references to other objects.

Figure 4-19 The color set object and its properties



These are the four accessible properties in a color set:

- **Color space.** The color space of all the color values in the color set. A color set can have only a single color space, which cannot be `gxIndexedSpace`.
- **Color-value array.** An array of color values (not `gxColor` structures). Only the types of color values specified in the `gxSetColor` union are valid in a color set.
- **Owner count.** The number of existing references to this color set object.
- **Tag list.** A list of references to custom information about this color set object, stored in private data structures called *tag objects*. The chapter "Tag Objects" in this book describes tag objects in general and how you can use them to add custom information to objects.

Colors and Color-Related Objects

QuickDraw GX provides functions to manipulate each of these properties. Note that there is no color profile property for a color set; profile information for the colors in a color set is found in the bitmap structure—or the color structure in the ink object—to which the color set is attached.

Color Values in a Color Set

The array of color values in a color set object can have up to 65,535 entries; each entry must be of one of the types defined in the `gxSetColor` union:

```
union gxSetColor{
    gxCMYKColor    cmyk;
    gxRGBColor     rgb;
    gxRGBAColor    rgba;
    gxHSVColor     hsv;
    gxHLSColor     hls;
    gxXYZColor     xyz;
    gxYXYColor     yxy;
    gxLUVColor     luv;
    gxLABColor     lab;
    gxYIQColor     yiq;
    gxColorValue   gray;
    gxGrayAColor   graya;
    unsigned short pixel16;
    unsigned long  pixel32;
    gxColorValue   component[4];
};
```

The `gxSetColor` union is an abbreviated color structure (see page 4-53). It has no profile or space fields, because individual colors within a color set cannot have different color spaces, and because the color profiles for the color values are defined elsewhere—in the individual colors, bitmaps, or transfer modes that use this color set. Also, the `gxSetColor` union has no `gxIndexedColor` field because color sets cannot be recursive (that is, colors in a color set cannot refer to colors in other color sets).

Default Color Sets

QuickDraw GX maintains several default color sets, one for each possible pixel size in bitmaps that use indexed space—1, 2, 4, and 8 bits. (Bitmaps with pixel sizes over 8 bits cannot use indexed space.) When you create a bitmap with a pixel size of 8 bits or less and specify `nil` for its color set, QuickDraw GX uses the appropriate default color set whenever you draw that bitmap.

Colors and Color-Related Objects

Each of the default color sets consists of a gray ramp, using color values in `gxGraySpace` that progress in order from white at an index value of 1 to black at the highest index value. For a pixel size of 1 bit, for example, the default color set consists of two colors: white and black.

You do not create a copy of any of the default color sets by calling the `GXNewColorSet` function; that function requires you to supply a specific array of color values.

You can inspect and change any of the default color sets by using the `GXGetDefaultColorSet` function, described on page 4-62, and the `GXSetDefaultColorSet` function, described on page 4-63. Bitmaps are described in the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

About Color Profile Objects

A color profile is a QuickDraw GX object that describes the color response of a specific device, class of device, or device configuration. As described in the section “Color Profiles” beginning on page 4-28, a color profile provides a quantitative description of a device’s color gamut in terms of standard, usually device-independent colors. QuickDraw GX uses color profiles for color matching when converting colors and when drawing shapes.

QuickDraw GX identifies a color profile object through a color profile *reference*. To obtain information about a color profile, you must send its reference as a parameter to a QuickDraw GX function (except that you can determine if two references identify the same color profile object simply by comparing them for equality, and you can examine a reference to see if it is `nil`).

Any QuickDraw GX color (`gxColor` structure), such as the color in a shape’s ink object, can include a reference to a color profile object. Any bitmap structure, including the bitmap in a view device object, can reference a color profile. (View devices are described in the chapter “View-Related Objects” in this book.) A transfer mode structure can also reference a color profile. If a color, bitmap, or transfer mode contains no specific reference to a color profile, QuickDraw GX uses a default profile when converting colors and when drawing.

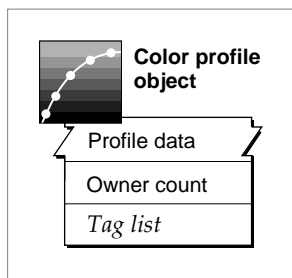
Even though color profiles are inherently device-specific, QuickDraw GX uses them consistently and performs color matching when needed. Most applications need not pay attention to color profiles or try to associate them with specific devices except when first creating colors. If you create a color, you should attach to it a color profile that describes the characteristics of the device on which the color was created. If the device’s characteristics are equivalent to the Apple 13-inch color monitor—or if you never need to display or print the color on another device—you need not attach a profile.

Color Profile Properties

The interface to color profile objects is entirely procedural. You manipulate the information in a color profile by modifying its properties using QuickDraw GX functions.

Color profile objects have three accessible properties, as shown in Figure 4-20. Note that, because a color profile is an object and not a data structure, the order of the properties as shown in Figure 4-20 is completely arbitrary. Properties in italics are references to other objects.

Figure 4-20 The color profile object and its properties



These are the three accessible properties in a color profile object:

- **Profile data.** Information specific to the individual profile, that usually includes color values and a set of data that plots the response of the device—from zero intensity to full intensity—when it generates each of the specified colors.
- **Owner count.** The number of existing references to this color profile object.
- **Tag list.** A list of references to custom information about this color profile object, stored in private data structures called *tag objects*. The chapter “Tag Objects” in this book describes tag objects in general and how you can use them to add custom information to objects.

QuickDraw GX provides functions to manipulate each of these properties.

Profile Data

The profile data is the actual color profile information, in the form of a ColorSync color profile structure. A QuickDraw GX color profile object is a wrapper for a ColorSync profile.

ColorSync profiles are specified by the `CMProfile` structure, which consists of the following parts:

- **Header.** A structure containing information such as the size, version, device type, and attributes of the profile. The header also contains the XYZ chromaticities of the device’s white point and black point, and an options field that specifies the type of color matching preferred (such as perceptual, colorimetric, or saturation matching).

Colors and Color-Related Objects

- **Profile chromaticities.** A structure that contains the XYZ chromaticities for the six primary and secondary colors (red, green, blue, cyan, magenta, yellow) at the limits of the device's gamut.
- **Profile response curves.** A variable-sized array of response curves for each of the primary and secondary colors, plus gray (plus black generation and undercolor removal for printer profiles).
- **Name string.** An international string, which consists of a Macintosh script code followed by a 63-byte text string, that identifies the profile. (Note that these are Macintosh script codes, which differ from QuickDraw GX script codes; Macintosh script codes are described in the Script Manager chapter of *Inside Macintosh: Text*.)
- **Custom data.** Information used by custom color-matching methods. It may include other kinds of color values or response curves.

The details of the `CMProfile` structure, including explanations of some of the terms used here, are given in the ColorSync Utilities chapter of *Inside Macintosh: Advanced Color Imaging*. All parts of the structure except for the custom data are accessible through ColorSync function calls. QuickDraw GX defines no structures or types for the profile data of a color profile object, although you can access the information if you know its format. See “Manipulating the Profile Data in a Color Profile Object” beginning on page 4-48.

The Default Color Profile

QuickDraw GX maintains a default color profile that it uses for color matching when no color profile is explicitly provided—that is, when the `profile` field of a color structure or bitmap structure is `nil`. The default color profile reflects the color response of the Apple 13-inch color monitor, as defined by ColorSync version 1.0.

You do not create a copy of the default color profile by calling the `GXNewColorProfile` function; that function requires you to supply profile data for the object you are creating. Also, you cannot change the characteristics of the default color profile object; there is no `GXSetDefaultColorProfile` function.

You can determine the actual profile chromaticities used for the default QuickDraw GX color profile by retrieving it and examining its profile data.

Zero-Length Profiles

QuickDraw GX automatically performs color matching whenever it draws or converts colors, and if you specify a `nil` color profile reference in any situation, QuickDraw GX uses the default color profile rather than using no profile.

In some cases, however, you may want to prevent color matching from occurring for individual colors or shapes, such as when comparing or calibrating different devices. To do so, you can use a zero-length profile. A **zero-length profile** is a color profile object in which the profile data is of zero length. It is a valid QuickDraw GX object—its reference is not `nil`—but it contains no data. If you attach a zero-length profile to a color, QuickDraw GX performs no matching when that color is drawn or converted.

Colors and Color-Related Objects

If, for example, you want to see how each attached device represents pure blue, you can specify pure blue for an ink's color, attach a zero-length profile to it, and draw a shape with that color to each device. If instead you specify `nil` for the profile when creating a color, QuickDraw GX matches the color, using the default color profile and each device's color profile, when drawing.

In the case of color conversions that require a profile (those between base families, such as from RGB to XYZ), QuickDraw GX uses the following conventions:

- If both profiles are zero-length, QuickDraw GX uses the default profile as both the source and the destination profile.
- If only one profile is zero-length, QuickDraw GX uses the other profile as both the source and the destination profile.

Note

To turn off color matching entirely when drawing to a view port, make sure that the `gxEnableMatchPort` attribute for that view port is cleared. (It is cleared by default.) View port attributes are discussed in the chapter “View-Related Devices” in this book. ♦

You can create a zero-length profile using the `GXNewColorProfile` function, described on page 4-79, or the `GXSetColorProfile` function, described on page 4-89.

Using Colors and Color-Related Objects

This section describes how to create and use colors, color sets, and color profile. It shows how you can

- assign colors to shapes and color profiles to colors
- test and compare colors
- convert colors from one color space to another, and apply color matching when converting and when scanning, displaying, or printing
- create and manipulate color set objects, to support indexed colors
- create and manipulate color profile objects, to support color matching

Assigning Colors to Shapes

Colors exist to affect the appearance of drawn shapes. QuickDraw GX shapes other than bitmaps and pictures get their color from the ink object that is part of the shape. One property of the ink object is color, a `gxColor` structure that describes the color of the associated shape.

Colors and Color-Related Objects

To assign or change a shape's color, therefore, you typically call the `GXSetInkColor` function for the ink associated with the shape whose color you are assigning. You can also call `GXSetShapeColor`, which performs the same task but allows you the convenience of specifying the shape object involved, rather than the ink object that actually contains the color information. (Conversely, to inspect the color of a shape, you call `GXGetInkColor` or `GXGetShapeColor`.) The `GXSetInkColor`, `GXSetShapeColor`, `GXGetInkColor`, and `GXGetShapeColor` functions are described in the chapter "Ink Objects" in this book.

Shapes that need more than one color are a special case. Bitmap shapes do not use the color information in their ink object. Instead, the color of each pixel in a bitmap shape is specified as a pixel value in the `gxBitmap` structure; depending on the storage size of each pixel, that pixel value may be an actual color value or it may be an index into a color set. To set the color of an individual pixel in a bitmap, you call the `GXSetShapePixel` function, specifying which pixel to modify and what its new color or new index value is. (Conversely, you can inspect the color of a pixel by calling `GXGetShapePixel`.)

Modifying the color values in a color set, as described in the section "Manipulating the Colors in a Color Set Object" on page 4-47, is another way to change the color or colors of a shape. In a bitmap using indexed color space, any pixels whose indexes refer to color values you have modified will be changed in appearance, even though their pixel values remain unchanged. You can use this technique to perform simple manipulations of a shape's colors.

Bitmap shapes, the `gxBitmap` structure, and the functions `GXSetShapePixel` and `GXGetShapePixel` are described in the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

Assigning Color Profiles to Colors

When the user creates or modifies shapes' colors, you assign colors to the ink objects or pixels associated with those shapes. To assure proper color matching, you can assign a color profile to each color or bitmap that the user creates. Normally, the user works with a monitor attached to the system; you can find the profile for that monitor by examining the `bitmap` property of the view device associated with the view port the user draws into. You can attach that profile to the user's colors. In the case of a single shape displayed on more than one device, you may have to pick (or allow the user to pick) which view device is the controlling one. In a `gxColor` structure or a `gxBitmap` structure, you place the color profile reference in the `profile` field.

If you assign no color profile to a color or bitmap, Quickdraw GX uses the default color profile when drawing or converting.

If you want to make sure that no matching occurs, assign a zero-length profile to the color or bitmap. A zero-length profile is a color profile object whose profile data is of zero length.

Comparing and Testing Colors

QuickDraw GX provides several functions that allow you to analyze individual color values for various purposes.

Checking for Out-of-Gamut Colors

If you have a color value that you want to test against a given color space or color set, you can use the `GXCheckColor` function. For example, you can use `GXCheckColor` to see if a given color is representable on a particular printer. If the color is not directly representable—that is, if it is *out of gamut*—you could alert the user to that fact. You could also call the `GXConvertColor` function to mimic the automatic color conversion that would take place in printing, to determine what color the printer would use to represent your given color.

Both `GXCheckColor` and `GXConvertColor` require the color space and color profile of the device the color is destined for. To get the color space and color profile of a printer, you can use the `GXGetPrinterViewDevice` and `GXGetViewDeviceBitmap` functions.

The `GXCheckColor` function is described on page 4-57. The `GXConvertColor` function is described on page 4-60. The `GXGetPrinterViewDevice` function is described in *Inside Macintosh: QuickDraw GX Printing*. The `GXGetViewDeviceBitmap` function is described in the chapter “View-Related Objects” in this book.

Checking Colors for Closeness and Color Space

If you want to compare a user-selected color with the range of colors in a color set, you can use the `GXGetColorDistance` function to determine how far the selected color is from any of the colors in the color set. If the selected color is close enough (in color-space distance) to one of the existing colors in the color set, you could call the `GXConvertColor` function to change the selected color to that closest color. Alternatively, you could call `GXGetColorSetParts` and `GXSetColorSetParts` to add the selected color to the color set or replace another color in the color set with the selected color.

As another example, suppose that you open a document containing shapes of various colors, and you want to save a grayscale version of that document. You might call `GXCheckColor` on each color in the document, and then `GXConvertColor` on each color whose color space is not already `gxGraySpace`. (You might also save the original color information as a tag object attached to each shape or ink, for later restoration.)

The `GXGetColorDistance` function is described on page 4-58. The `GXConvertColor` function is described on page 4-60. The `GXGetColorSetParts` function is described on page 4-75; the `GXSetColorSetParts` function is described on page 4-76. The `GXCheckColor` function is described on page 4-57.

Predicting Drawing Results

You can preflight, or predict, the results of a drawing operation by using the `GXCombineColor` function. You supply a destination color, and `GXCombineColor` tells you what would happen if a shape using the ink object you specify were drawn to a destination of that color. This function is as much a test of transfer mode as it is of source and destination colors; you can use it to see how, or even if, drawing occurs under the conditions you specify. For example, if you are using the `gxMigrateMode` transfer mode, you may want to adjust the operand so that the result color exactly equals the source color for a particular destination color. You can call `GXCombineColor` with different operand values until you get the result you want, and then draw the actual shape. Transfer modes, operands, and source and destination colors are described in the chapter “Ink Objects” in this book.

The `GXCombineColor` function is described on page 4-59.

Converting and Matching Colors

Although conversion among color spaces happens automatically whenever necessary during the drawing process, you can also explicitly convert colors if you need to. The following code fragment uses `GXConvertColor` to modify the hue of a shape, preserving its luminance and saturation. Such a technique is one way to perform color animation. The code gets the color of shape `theShape`, converts it to HSV space, increases the hue value just enough to make a perceptible difference, reassigns the color to the shape, and draws the shape:

```
gxColor      oldColor;
GXGetShapeColor(theShape, &oldColor);
GXConvertColor(&oldColor, hsvSpace, nil, nil);
oldColor.element.hsv.hue += 0x300;
GXSetShapeColor(theShape, &oldColor);
GXDrawShape(theShape);
```

(Note that the final `nil` parameter in the call to `GXConvertColor` means that the converted color reassigned to the shape uses the default color profile, whether or not the original one did.)

Color matching happens automatically whenever you draw a shape or convert colors. If the profile reference in a color is `nil`, color correction still occurs when needed, as when converting from RGB to CMYK color space. In those cases, the default profile is used.

In some cases, you may want to prevent color matching from occurring for an individual color, such as when comparing or calibrating different devices. If you attach a zero-length profile to a color, QuickDraw GX performs no matching when that color is drawn or converted to another color space.

To prevent color matching from occurring during all drawing to a given view port, clear the `gxEnableMatchPort` attribute of that view port. Note that, because color matching can slow down the drawing process, this attribute is cleared by default on all view ports.

Colors and Color-Related Objects

Therefore, if you want color matching to occur when drawing to the screen, you must explicitly set `gxEnableMatchPort`. Even if you do want matching to occur, you might still clear `gxEnableMatchPort` temporarily during scrolling or other repetitive drawing processes. (For printing, QuickDraw GX automatically takes care of making sure that color matching occurs when it is needed.)

If you want to specify a particular kind of color-matching method other than the one specified in the profile attached to the color you are matching, your application can either modify the information in the color profile object using QuickDraw GX calls, or make calls to the ColorSync Utilities to specify the one you want.

To allow the user to preview on the screen what printing would look like, you can mimic on the monitor the profile characteristics of the printer. You need to convert the color you are drawing to the color space of the printer—applying the printer’s color profile—and then convert that color back to the monitor’s color space—applying the monitor’s color profile—and then draw. One way to do that is to create an offscreen view group with the printer’s color space and color profile, and draw into a view port in that view group. Then, draw from the bitmap of the offscreen view port into the view port of the monitor.

Color matching is discussed in the section “Color Conversion and Color Matching” beginning on page 4-26. Color profiles, the default profile, and zero-length profiles are discussed in the section “About Color Profile Objects” beginning on page 4-35. The `gxEnableMatchPort` view port attribute is described in the chapter “View-Related Devices” in this book.

Creating and Manipulating Color Set and Color Profile Objects

This section describes how you can create and interact with color set objects and color profile objects as whole entities—to create, dispose of, copy, compare, clone, load, and unload them. Because color sets and color profiles are QuickDraw GX objects, and you use similar sets of functions to manipulate them, they are considered together in each of the subsequent sections. Manipulating the individual properties of color sets and color profiles is described under “Manipulating Object Properties of Color Sets and Color Profiles” beginning on page 4-46.

Creating and Disposing of a Color Set or Color Profile

QuickDraw GX provides the `GXNewColorSet` function to allow you to create a new color set. You can also create a new color set that is a copy of an existing color set by calling `GXCopyToColorSet`.

Once you have created a color set object, you can attach it to a color structure, bitmap structure, or transfer mode structure by putting a reference to it in the color or bitmap or transfer mode. Colors, bitmaps, and transfer modes also include a specification of the color space they use; if they use a color set, they must use `gxIndexedSpace` for their color space.

Colors and Color-Related Objects

The following code fragment creates a simple color set (`theSet`) with two RGB colors: black and white. It assigns the color set to the bitmap structure `theBits`, which it then assigns to the bitmap shape `theBitmap`, which it finally assigns to the view device `theDevice`. The code then disposes of the color set and bitmap shape since those references are no longer needed:

```
gxSetColor  theColors[2];
gxSetColor  *pColor;
gxColorSet  theSet;
gxBitmap    theBits
gxShape     theBitmap

.
.  /* initialize theBits and theBitmap (not shown) */
.
pColor = &theColors[0];
pColor->rgb.red = pColor->rgb.green =
                pColor->rgb.blue = gxColorValue1;
pColor++;
pColor->rgb.red = pColor->rgb.green = pColor->rgb.blue = 0x0000;

theSet = GXNewColorSet(gxRGBSpace, 2, theColors);
theBits.set = theSet;
GXSetBitmap(theBitmap, &theBits, nil);
GXSetViewDeviceBitmap(theDevice, theBitmap);
GXDisposeColorSet(theSet);
GXDisposeShape(theBitmap);
```

Note

If you use `GXNewColorSet` to create a color set, and then assign it as a default color set with `GXSetDefaultColorSet`, be sure to dispose of your reference to that color set immediately after assigning it as the default. That way the new default color set will have the proper owner count of 1, as required by QuickDraw GX. ♦

For color profile objects, QuickDraw GX provides the `GXNewColorProfile` function (and the `GXCopyToColorProfile` function) to allow you to create new color profiles. If you have profile information that you want to attach to a color or to a bitmap, you can put that information in object form with `GXNewColorProfile` and attach it (by reference) to the color or bitmap. For simple drawing, you typically never have to do this, but you might want to create a color profile object in these special instances:

- If you want to inhibit color matching for a particular color, you can create a zero-length profile (one with no profile data) and attach it to the color.
- If you have access to a profile structure, either as a resource or through calls to the ColorSync Utilities, you can turn that structure into a QuickDraw GX color profile by creating a color profile object with that structure as the profile data.

Colors and Color-Related Objects

- If your application is a scanning application, it can create a color profile object from information in the scanner's driver and attach that profile to the bitmap shapes it creates.
- If your application is a calibration program that develops profile information for a device, it can create a color profile object to hold the profile information it generates during the calibration process and to display the results to the operator of the calibration program.

If your program is a device driver, it contains profile information in the form of color profile resources; it does not need to create color profile objects. How device drivers store color profile information is described in the printing resources chapter of *Inside Macintosh: QuickDraw GX Printing Extensions and Printer Drivers*.

To delete your application's reference to a color set or color profile object, call the `GXDisposeColorSet` or `GXDisposeColorProfile` function. Calling either function may or may not actually release the memory allocated for the object, depending on the object's owner count. Both of these functions decrease the owner count of the color set or color profile by 1; if that brings the owner count to zero, the object is completely deleted and its memory released. See "Manipulating Owner Counts" on page 4-46.

The `GXNewColorSet` function is described on page 4-64; the `GXNewColorProfile` function is described on page 4-79. The `GXDisposeColorSet` function is described on page 4-65; the `GXDisposeColorProfile` function is described on page 4-80.

Copying, Comparing, and Cloning Color Sets and Color Profiles

You can use the `GXCopyToColorSet` function to copy color information from one color set object to another or to create a new copy of an existing color set. You can use the `GXCopyToColorProfile` function to copy profile information from one color profile object to another or to create a new copy of an existing color profile.

You can test if two references refer to the same color set or color profile object by simply comparing the references for equality. You can also test two different color set or color profile objects for equality with the `GXEqualColorSet` and `GXEqualColorProfile` functions, respectively. For two color sets to be equal, their color spaces and colors must be identical; for two color profiles to be equal, their profile information and their attributes must be equal. In either case, the common object properties (owner count and tag list) do not need to be identical for the objects to be considered equal.

Object copies created with the `GXCopyToColorSet` and `GXCopyToColorProfile` functions are always equal, in terms of the criteria just listed, to the objects from which they were copied.

In certain circumstances, you may want to copy a reference to a color set or color profile without actually copying the object. For example, you may want two variables to refer to the same color set or color profile object, so that altering one of them affects both. This is called **cloning** an object, rather than copying it. You can use the `GXCloneColorSet` and `GXCloneColorProfile` functions to clone a color set or color profile, respectively.

Colors and Color-Related Objects

Functionally, `GXCloneColorSet` and `GXCloneColorProfile` do nothing more than increase the owner count of the specified object. For more information about cloning objects, see the chapter “Introduction to Objects” in this book. For information on manipulating owner counts, see the section “Manipulating Owner Counts” on page 4-46.

The following code fragment initializes a bitmap structure to be used for offscreen drawing, assigns a color set object (`commonColorSet`) to it, and then creates a bitmap shape (`shMap`) with that bitmap. The code, for its own purposes of tracking owner count (not shown here), clones the color set rather than just assigning it to the bitmap shape. In general, cloning is not necessary when you assign a color set to a bitmap, because when you then call `GXNewBitmap` to create the bitmap shape (as this code fragment does), QuickDraw GX increases the color set’s owner count for you.

```
gxBitmap      map;
gxPoint       pt = {0, 0};
gxShape       shMap = nil;
.
.  /* set the bitmap's width, height, and pixel size */
.
map.rowBytes = 0L;
map.image = nil;
map.space = gxIndexedSpace;
map.profile = nil;
map.set = GXCloneColorSet(commonColorSet);
shMap = GXNewBitmap(&map, &pt);
```

QuickDraw GX will decrease the owner count of the color set when the shape `shMap` is disposed of, but the application code will also need to call `GXDisposeColorSet` at some point, to balance the `GXCloneColorSet` call it makes here.

The `GXCopyToColorSet` function is described on page 4-66; the `GXCopyToColorProfile` function is described on page 4-81. The `GXEqualColorSet` function is described on page 4-67; the `GXEqualColorProfile` function is described on page 4-82. The `GXCloneColorSet` function is described on page 4-68; the `GXCloneColorProfile` function is described on page 4-83.

Loading and Unloading Color Sets and Color Profiles

Although you rarely need to, you can influence memory-allocation decisions involving objects that you have created. If your application needs to have a color set object or color profile object in memory, it can force QuickDraw GX to load the object into memory. When your application no longer needs the color set or color profile in a loaded state, it can instruct QuickDraw GX to unload the object.

Colors and Color-Related Objects

You call the `GXLoadColorSet` or `GXLoadColorProfile` function to make sure that a color set or color profile object is in memory; if it has been unloaded, QuickDraw GX brings it into memory. You can call the `GXUnloadColorSet` or `GXUnloadColorProfile` function to instruct QuickDraw GX that it is free to unload the color set or color profile at any time. These functions are described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Manipulating Object Properties of Color Sets and Color Profiles

This section describes how to manipulate the common object properties of color sets and color profiles: owner count and tag list. It also describes how to manipulate the colors of a color set and the profile data of a color profile.

For manipulating color sets and color profiles as whole objects, see “Creating and Manipulating Color Set and Color Profile Objects” beginning on page 4-42.

Manipulating Owner Counts

The owner count of an object indicates the number of current references to that object. In general, QuickDraw GX manages owner counts for you. For example, when you create a new color set object, QuickDraw GX sets the owner count of the new color set to 1. If you assign that color profile object to a bitmap structure and then assign that bitmap structure to a bitmap shape, QuickDraw GX increments the color profile’s owner count, corresponding to the new reference to the color profile contained in the bitmap structure.

In some situations, as when switching color profiles or color sets among objects that reference them, you may want to directly manage their owner counts yourself. To do so, you can

- use the functions `GXGetColorSetOwners` or `GXGetColorProfileOwners` to determine the current owner count
- use the functions `GXCloneColorSet` or `GXCloneColorProfile` to increment the owner count whenever you create a new reference to the object
- use the functions `GXDisposeColorSet` or `GXDisposeColorProfile` to decrement the owner count, freeing the memory used by the color set or color profile if the owner count goes to 0

The code fragment on page 4-45 shows an example of an application explicitly managing the owner count of a color profile object.

The `GXGetColorSetOwners` function is described on page 4-69. The `GXGetColorProfileOwners` function is described on page 4-84.

In the chapter “Style Objects” in this book, the section on manipulating a style object’s owner count discusses two common owner-count problems and how to avoid them. The problems are discussed in terms of style objects, but they apply equally well to color sets and color profiles. Refer to that discussion if you find that the color-related objects you create have owner counts that are higher or lower than you expect.

Getting and Setting Tag References

You can examine the list of references to tag objects currently associated with a color set object or color profile object by using the `GXGetColorSetTags` or `GXGetColorProfileTags` function. Once you create a tag object, you can attach it to its object using the `GXSetColorSetTags` or `GXSetColorProfileTags` function. You can attach as many tag objects as you like to a color set or color profile.

Tag objects and the basic functions for manipulating them are described in the chapter “Tag Objects” in this book. That chapter also lists the common tag types defined and reserved by Apple Computer, Inc.

The `GXGetColorSetTags` function is described on page 4-70; the `GXGetColorProfileTags` function is described on page 4-85. The `GXSetColorSetTags` function is described on page 4-71; the `GXSetColorProfileTags` function is described on page 4-86.

Manipulating the Colors in a Color Set Object

If you are using indexed color space, you can gain access to the array of colors in the space’s color set or to any contiguous subset of the colors in the array. You can then inspect, rearrange, modify, or add or delete colors from the array.

For example, suppose you want to sort the colors in a color set so that they will display in a visually useful manner in a palette for the user. You could first call the `GXGetColorSet` function to get the array of colors. You could then sort the colors (say, by hue (H) in `gxHSVSpace`), and then return the array to the color set by calling the `GXSetColorSet` function.

Alternatively, suppose you already have a luminance-sorted array of colors in a color set, and you want to convert the first (darkest) color in the array to pure black. Instead of accessing the entire array, you can call `GXGetColorSetParts` to get only the first color in the array. You can then change that color to black, and reinsert it in the color set by calling `GXSetColorSetParts`.

To add colors to or delete colors from a color set, call `GXGetColorSet`, modify the color-value array as needed, and then call `GXSetColorSet` to place the new array in the color set.

To change the color space of a color set, follow this sequence of calls:

- Call `GXGetColorSet` to obtain the color-value array.
- Call `GXConvertColor` on each color value in the array to convert the individual color values from one space to the other.
- Call `GXSetColorSet` to place the same array in the color set, but with a different value specified for the color space.

Remember that simply changing the color space of a color set does not convert the individual color values from one space to the other.

Colors and Color-Related Objects

As an example of color-set manipulation, the following code fragment from a drawing routine matches each of the colors of a color set used by the shape `matchShape` to a specific color profile (`qmsProfile`). The code uses the `GXGetColorSet` function to fill out a temporary array of color values (`mycolors`) from the color set, converts each color (from RGB space with a `nil` profile to RGB space with `qmsProfile`, in this case) with the `GXConvertColor` function, and then reassigns the color values to the color set with the `GXSetColorSet` function.

```
gxSetColor mycolors[256];
oldColorCount = GXGetColorSet(GXGetShapeColorSet(matchShape),
                              nil, mycolors);
for (i = 0; i < oldColorCount; i++)
{
    gxColor tmpColor;
    tmpColor.space = gxRGBSpace;
    tmpColor.profile = nil;
    tmpColor.element.rgb = mycolors[i].rgb;
    GXConvertColor(&tmpColor, gxRGBSpace, nil, qmsProfile);
    mycolors[i].rgb = tmpColor.element.rgb;
}
GXSetColorSet(GXGetShapeColorSet(matchShape), gxRGBSpace,
              oldColorCount, mycolors);
```

The `GXGetColorSet` function is described on page 4-73. The `GXSetColorSet` function is described on page 4-74. The `GXGetColorSetParts` function is described on page 4-75. The `GXSetColorSetParts` function is described on page 4-76.

Manipulating the Profile Data in a Color Profile Object

QuickDraw GX defines no structures or types for the profile data of a color profile object. For drawing or converting colors, most applications have no need to access or alter the data in a color profile object. For special needs, however, such as changing the type of match you want to perform, using a custom color-matching method, or inspecting the name of a profile, you can—with knowledge of the details of the `CMPProfile` structure—access and alter the profile data of an existing color profile object. Also, if your application is a calibration program that creates color profiles for devices, or if it is an imaging application that allows users to customize color profiles for specific uses, you need access to profile information in order to make or modify a color profile object.

One way to do this is to use ColorSync functions to manipulate a ColorSync profile directly, and then use the QuickDraw GX function `GXNewColorProfile` to convert it to a color profile object. ColorSync profiles are commonly in the ColorSync profiles folder on the user's system, and ColorSync can provide you with a list of those profiles.

Colors and Color-Related Objects

More directly, you can call the `GXGetColorProfile` function to obtain the profile data for a given profile. Knowing the structure of a ColorSync color profile, you can then modify that information as needed, and return the altered data to the color profile object by calling the `GXSetColorProfile` function.

Note

If you alter the header of a ColorSync color profile to specify a particular color space in the `dataType` field, and then apply that profile to a color defined in terms of a different color space, QuickDraw GX ignores the new header data and specifies the color space implied by the color value you pass to the profile. ♦

Yet another approach is to directly modify the profile data of a color profile object in place, in QuickDraw GX memory. First, you call the `GXLockColorProfile` function to prevent the profile data from being relocated, and then you call `GXGetColorProfileStructure` to get a pointer to the profile data. After manipulating the data, you must call `GXUnlockColorProfile` to release the data for relocation. Remember that you cannot change the *size* of the profile data with these calls, only its contents; if your manipulations require a change in the size of the data, you must use `GXGetColorProfile` and `GXSetColorProfile`.

IMPORTANT

Memory-handling complications can occur with locked objects. Locking an object fragments the QuickDraw GX heap, which can result in lower performance. Furthermore, if a fragmented-memory condition occurs during a call, QuickDraw GX may unlock all objects and restart the call. Therefore, be careful about performing memory-intensive operations while there are locked objects in QuickDraw GX memory; they may become unlocked without warning. ▲

The `GXNewColorProfile` function is described on page 4-79. The `GXGetColorProfile` function is described on page 4-88. The `GXSetColorProfile` function is described on page 4-89. The `GXLockColorProfile` function is described on page 4-90. The `GXGetColorProfileStructure` function is described on page 4-92. The `GXUnlockColorProfile` function is described on page 4-91.

Colors and Color-Related Objects Reference

This section provides reference information to the data structures and functions that allow you to work with colors and create and manipulate color sets and color profiles, and to alter their properties. It describes

- the constants and data types that define colors and color-related objects
- the QuickDraw GX functions that operate on colors
- the QuickDraw GX functions that operate on color sets
- the QuickDraw GX functions that operate on color profiles

Constants and Data Types

This section describes the constants and data types that define

- colors and color spaces
- color set objects
- color profile objects

Color-Component Values

Each color component in a color space (other than an indexed color space) is described by a numeric color-component value, defined by the `gxColorValue` type definition:

```
typedef unsigned short gxColorValue;
```

Color-component values can vary from 0 (no intensity) to 0xFFFF (maximum intensity). You can use the constant `gxColorValue1` to represent 0xFFFF.

Color Values

Color-component values combine to form color values. Each color value is a complete specification of a single color in a given color space. QuickDraw GX recognizes the following ten fundamental types of color values:

- CMYK color value. It contains color-component values for cyan, magenta, yellow, and black. It is defined by the `gxCMYKColor` type definition:

```
struct gxCMYKColor{
    gxColorValue    cyan;
    gxColorValue    magenta;
    gxColorValue    yellow;
    gxColorValue    black;
};
```

- RGB color value. It contains color-component values for red, green, and blue. It is defined by the `gxRGBColor` type definition:

```
struct gxRGBColor{
    gxColorValue    red;
    gxColorValue    green;
    gxColorValue    blue;
};
```

Colors and Color-Related Objects

- Alpha-channel RGB color value. It contains color-component values for red, green, and blue, plus a fourth (alpha) color-component value representing opacity. It is defined by the `gxRGBAColor` type definition:

```
struct gxRGBAColor{
    gxColorValue    red;
    gxColorValue    green;
    gxColorValue    blue;
    gxColorValue    alpha;
};
```

- HSV color value. It contains color-component values for hue, saturation, and value. It is defined by the `gxHSVColor` type definition:

```
struct gxHSVColor{
    gxColorValue    hue;
    gxColorValue    saturation;
    gxColorValue    value;
};
```

- HLS color value. It contains color-component values for hue, lightness, and saturation. It is defined by the `gxHLSColor` type definition:

```
struct gxHLSColor{
    gxColorValue    hue;
    gxColorValue    lightness;
    gxColorValue    saturation;
};
```

- XYZ color value. It contains color-component values for the X, Y, and Z tristimulus values. It is defined by the `gxXYZColor` type definition:

```
struct gxXYZColor {
    gxColorValue    x;
    gxColorValue    y;
    gxColorValue    z;
};
```

- Yxy color value. It contains color-component values for the Y, x, and y chromaticity axes. (Note that the Y component is identified in this color structure as `capY`.) It is defined by the `gxYXYColor` type definition:

```
struct gxYXYColor {
    gxColorValue    capY;
    gxColorValue    x;
    gxColorValue    y;
};
```

Colors and Color-Related Objects

- L*u*v* color value. It contains color-component values for the L*, u*, and v* axes. It is defined by the `gxLUVColor` type definition:

```
struct gxLUVColor {
    gxColorValue  l;
    gxColorValue  u;
    gxColorValue  v;
};
```

- L*a*b* color value. It contains color-component values for the L*, a*, and b* axes. It is defined by the `gxLABColor` type definition:

```
struct gxLABColor {
    gxColorValue  l;
    gxColorValue  a;
    gxColorValue  b;
};
```

- YIQ color value. It contains color-component values for the Y, I, and Q axes. It is defined by the `gxYIQColor` type definition:

```
struct gxYIQColor{
    gxColorValue  y;
    gxColorValue  i;
    gxColorValue  q;
};
```

- Grayscale color value. It contains a single color-component value for luminance.
- Alpha-channel grayscale color value, containing a color-component value for luminance, plus a second (alpha) color-component value representing opacity. It is defined by the `gxGrayAColor` type definition:

```
struct gxGrayAColor{
    gxColorValue  gray;
    gxColorValue  alpha;
};
```

- Indexed color value. It contains an index value (of type `gxColorIndex`) and a reference to a color set object. The color is obtained by using the index value as an offset into the color set. Indexed color is defined by the `gxIndexedColor` type definition:

```
typedef long gxColorIndex;

struct gxIndexedColor{
    gxColorIndex  index;
    gxColorSet    set;
};
```


The Color Structure

A color value, plus a specification of the color space it belongs to, plus an optional reference to a color profile to use for color matching, constitute a color in QuickDraw GX. A color is a structure defined by the `gxColor` type definition:

```
struct gxColor{
    gxColorSpace      space;
    gxColorProfile     profile;
    union {
        struct gxCMYKColor    cmyk;
        struct gxRGBColor     rgb;
        struct gxRGBAColor    rgba;
        struct gxHSVColor     hsv;
        struct gxHLSColor     hls;
        struct gxXYZColor     xyz;
        struct gxYXYColor     yxy;
        struct gxLUVColor     luv;
        struct gxLABColor     lab;
        struct gxYIQColor     yiq;
        gxColorValue          gray;
        struct gxGrayAColor    graya;
        unsigned short         pixel16;
        unsigned long          pixel32;
        struct gxIndexedColor  indexed;
        gxColorValue           component[4];
    } element;
};
```

Field descriptions

space	The color space for this color.
profile	A reference to a color profile to be used for color matching when drawing or when converting this color to another color space. If this field is <code>nil</code> , the default QuickDraw GX color profile is used for matching.
element	The color value for this color.

The `element` field is a union that can contain any one of the following fields:

cmyk	A CMYK color value.
rgb	An RGB color value.
rgba	An alpha-channel RGB color value.
hsv	An HSV color value.
hls	An HLS color value.
xyz	An XYZ color value.
yxy	A Yxy color value.

Colors and Color-Related Objects

luv	An L*u*v* color value.
lab	An L*a*b* color value.
yiiq	A YIQ color value.
gray	A grayscale color value
graya	An alpha-channel grayscale color value.
pixel16	A 16-bit pixel value, in gxRGB16Space format.
pixel32	A 32-bit pixel value, in any of the 32-bit color space formats.
indexed	An indexed color value.
component	An array of 4 undefined color-component values. Useful for indexing through the color one component at a time, as when working with different transfer modes for each color component.

Color Packing

You can store color values according to their standard definitions, or in packed format to save space. QuickDraw GX recognizes six kinds of color-value storage, defined in the `gxColorPackingTypes` enumeration:

```
enum gxColorPackingTypes{
    gxNoColorPacking      = 0x0000,
    gxAlphaSpace          = 0x0080,
    gxWord5ColorPacking   = 0x0500,
    gxLong8ColorPacking   = 0x0800,
    gxLong10ColorPacking  = 0x0a00,
    gxAlphaFirstPacking   = 0x1000
};
```

Constant descriptions

<code>gxNoColorPacking</code>	No packing applied; colors are stored with 16 bits per component.
<code>gxAlphaSpace</code>	An alpha channel is included in the color description. The alpha component follows the other components in storage.
<code>gxWord5ColorPacking</code>	Colors are stored with 5 bits per component. Unused bits in the storage space are the high-order bits.
<code>gxLong8ColorPacking</code>	Colors are stored with 8 bits per component. Unused bits in the storage space are the high-order bits.
<code>gxLong10ColorPacking</code>	Colors are stored with 10 bits per component. Unused bits in the storage space are the high-order bits.
<code>gxAlphaFirstPacking</code>	An alpha channel is included in the color description. The alpha component precedes the other components in storage.

Colors and Color-Related Objects

The color-packing values are flags that are added to color-space definitions to define different kinds of packed color spaces. Note that the specification of an alpha channel in a color space is achieved with a color-packing flag. To see how these values are applied to the definitions of color spaces, see the section “Color Spaces,” next.

When QuickDraw GX converts from an unpacked color space to a packed color space, the color-component values are truncated (low-order bits lost) to fit the packed format. When QuickDraw GX converts from a packed color space to an unpacked color space, the color-component values are shifted leftward (padded with zeros in the low-order bits) to fit the unpacked format.

Color Spaces

A color space defines how a color value is represented. Each color space specifies the number, order, and size of the color-component values that make up a color value in that space. QuickDraw GX recognizes 31 color spaces, defined in the `gxColorSpace` enumeration:

```
enum gxColorSpaces{
    gxNoSpace          = 0,
    gxRGBSpace,
    gxCMYKSpace,
    gxHSVSpace,
    gxHLSpace,
    gxXYXSpace,
    gxXYZSpace,
    gxLUVSpace,
    gxLABSpace,
    gxYIQSpace,
    gxNTSCSpace        = gxYIQSpace,
    gxPALSpace          = gxYIQSpace,
    gxGraySpace,
    gxIndexedSpace,
    gxRGBASpace         = gxRGBSpace + gxAlphaSpace,
    gxGrayASpace        = gxGraySpace + gxAlphaSpace,
    gxRGB16Space        = gxWord5ColorPacking + gxRGBSpace,
    gxRGB32Space        = gxLong8ColorPacking + gxRGBSpace,
    gxARGB32Space       = gxLong8ColorPacking + gxAlphaFirstPacking
                        + gxRGBASpace,
    gxCMYK32Space       = gxLong8ColorPacking + gxCMYKSpace,
    gxHSV32Space        = gxLong10ColorPacking + gxHSVSpace,
    gxHLS32Space        = gxLong10ColorPacking + gxHLSpace,
    gxXYX32Space        = gxLong10ColorPacking + gxXYXSpace,
    gxXYZ32Space        = gxLong10ColorPacking + gxXYZSpace,
    gxLUV32Space        = gxLong10ColorPacking + gxLUVSpace,
```

Colors and Color-Related Objects

```

    gxLAB32Space    = gxLong10ColorPacking + gxLABSpace,
    gxYIQ32Space    = gxLong10ColorPacking + gxYIQSpace,
    gxNTSC32Space   = gxYIQ32Space,
    gxPAL32Space    = gxYIQ32Space,
};

```

```
typedef long gxColorSpace;
```

Note that color spaces from `gxRGBASpace` through `gxYIQ32Space` use color-packing flags in their definitions. Those flags are described in the previous section, “Color Packing.”

The individual color spaces are described in the section “Color Spaces” beginning on page 4-6.

The Color Set Object

QuickDraw GX provides you with access to an individual color set object through a `gxColorSet` reference:

```
typedef struct gxPrivateColorSetRecord *gxColorSet;
```

In this type definition, `gxColorSet` is a type-checked reference, not an actual pointer to any defined structure. The contents of the color set object are private.

The gxSetColor Union

A color set object is essentially an array of color values. The acceptable types of color values that it may contain are defined by the `gxSetColor` union:

```

union gxSetColor{
    gxCMYKColor    cmyk;
    gxRGBColor     rgb;
    gxRGBAColor    rgba;
    gxHSVColor     hsv;
    gxHLSColor     hls;
    gxXYZColor     xyz;
    gxYXYColor     yxy;
    gxLUVColor     luv;
    gxLABColor     lab;
    gxYIQColor     yiq;
    gxColorValue   gray;
    gxGrayAColor   graya;
    unsigned short pixel16;
    unsigned long  pixel32;
    gxColorValue   component[4];
};

```

Colors and Color-Related Objects

The `gxSetColor` union is an abbreviated `gxColor` structure. The `gxColor` structure is described on page 4-53.

The Color Profile Object

A color profile describes how to match a color with the colors in a color space. QuickDraw GX provides you with access to an individual color profile object through a `gxColorProfile` reference:

```
typedef struct gxPrivateProfileRecord *gxColorProfile;
```

In this type definition, `gxColorProfile` is a type-checked reference, not an actual pointer to any defined structure. The contents of the color profile object are private.

Color profile objects contain color profile structures as defined by the Macintosh ColorSync Utilities. See *Inside Macintosh: Advanced Color Imaging* for more information.

Color Functions

The functions in this section manipulate color structures, allowing you to test a color, compare two colors, combine two colors, and convert a color from one color space to another. Colors are described in the section “Color-Component Values, Color Values, and Colors” beginning on page 4-25. The color structure (type `gxColor`) is described on page 4-53.

GXCheckColor

You can use the `GXCheckColor` function to determine if a color is either within a given gamut in a particular color space, or representable in a given color set.

```
boolean GXCheckColor(const gxColor *source, gxColorSpace space,
                    gxColorSet aSet, gxColorProfile profile);
```

<code>source</code>	A pointer to the color to check.
<code>space</code>	The color space to check the source color against.
<code>aSet</code>	A reference to a color set to check the source color against. This parameter must be <code>nil</code> if the <code>space</code> parameter is not <code>gxIndexedSpace</code> .
<code>profile</code>	A reference to a color profile to check the source color against. <code>GXCheckColor</code> determines whether the source color is within the color gamut represented by this profile and the space color space.

function result `true` if the source color is contained in the specified color set, or if it is within the gamut of the specified color space and color profile; otherwise, `false`.

Colors and Color-Related Objects

DESCRIPTION

The `GXCheckColor` function has two purposes. One is that you can use it to see if a given color exactly matches a color within a color set. For example, you can test whether a color matches a Pantone® or other spot color standard. To do this check, make sure that the `space` parameter specifies indexed color space and that the `aSet` parameter is not `nil`.

You can also use the `GXCheckColor` function to see if a given color can be drawn on a given view device. The function converts the source color to the color space represented in the `space` parameter, using the color profile in the `profile` parameter. If the resulting color is out of the gamut represented by `space` and `profile`, the function returns `false`.

SPECIAL CONSIDERATIONS

If you are using this function to test a color against a color set, it is unlikely to find a match (which must be exact) unless the source color and the color set referenced in the `aSet` parameter are based on the same color space and use identical color profiles.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`color_is_nil`
`colorSpace_out_of_range` (debugging version)

Warnings

`colorSet_index_out_of_range`

GXGetColorDistance

You can use the `GXGetColorDistance` function to determine the color-space distance between two colors.

```
Fixed GXGetColorDistance(const gxColor *target,
                        const gxColor *source);
```

`target` A pointer to the target color.
`source` A pointer to the source color.

function result The color-space distance between the two colors.

Colors and Color-Related Objects

DESCRIPTION

The `GXGetColorDistance` function is useful in colorimetric applications and for judging perceived closeness of colors. It calculates how similar two colors are by determining the color-space distance between them. The distance calculation is performed in the color space of the target color. If the two colors are not in the same space, `GXGetColorDistance` converts the source color to the target color space before calculating the distance.

If the target color space is `gxIndexedSpace`, `GXGetColorDistance` uses the color space of the target color set.

The distance formula used is the standard Euclidean distance:

```
distance = Sqrt( (b0-a0)^2 + (b1-a1)^2 + ... );
```

SPECIAL CONSIDERATIONS

Because some of the color spaces are not linear, distances calculated in one space are not necessarily proportional to distances calculated in another space.

ERRORS, WARNINGS, AND NOTICES**Errors**

```
out_of_memory
color_is_nil
colorSpace_out_of_range          (debugging version)
```

Warnings

```
colorSet_index_out_of_range
```

GXCombineColor

You can use the `GXCombineColor` function to combine two colors with a transfer mode to get a result color for testing, without actually drawing.

```
gxColor *GXCombineColor(gxColor *target, gxInk operand);
```

target A pointer to the target color, which represents the destination color for drawing. On return, `target` points to the result color.

operand A reference to an ink object, which represents the source color and the transfer mode for drawing.

function result A pointer to the result color.

Colors and Color-Related Objects

DESCRIPTION

The `GXCombineColor` function lets you preview or predict the results of drawing without actually carrying out a drawing operation. The function applies the color and transfer mode of the ink object referenced in the `operand` parameter to the color specified in the `target` parameter. It calculates the result of drawing, with the ink's color as the source color and the target color as the destination color.

`GXCombineColor` modifies the target color to reflect the operation and also returns a pointer to the resulting color. If `target` or `operand` is `nil`, the function posts an error and returns `nil`.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`ink_is_nil`
`color_is_nil`
`colorSpace_out_of_range` (debugging version)
`invalid_transferMode_colorSpace` (debugging version)

Warnings

`colorSet_index_out_of_range`

SEE ALSO

Ink objects and transfer modes for drawing are described in the chapter “Ink Objects” in this book.

GXConvertColor

You can use the `GXConvertColor` function to convert a color from one color space to another.

```
gxColor *GXConvertColor(gxColor *target, gxColorSpace space,
                        gxColorSet aSet, gxColorProfile profile);
```

<code>target</code>	A pointer to the color to be converted. On return, <code>target</code> points to the converted color.
<code>space</code>	The color space to convert the target color to.
<code>aSet</code>	A reference to the color set to assign to the color space of the target color. This parameter must be <code>nil</code> if the <code>space</code> parameter is not <code>gxIndexedSpace</code> .
<code>profile</code>	A reference to the color profile to assign to the converted color (that is, to use as the destination profile for the conversion). If you pass <code>nil</code> for this parameter, QuickDraw GX uses the default color profile.

function result A pointer to the converted color.

Colors and Color-Related Objects

DESCRIPTION

The `GXConvertColor` function converts a color from one color space to another. The target color is both the input and the output color for this function; the function modifies the target color to reflect the conversion and also returns a pointer to the converted color. If `target` is `nil`, the function posts an error and returns `nil`.

If appropriate, `GXConvertColor` automatically performs color matching when converting the color. The color profile—if any—associated with the target color is used to correct the input color, and the color profile referenced in the `profile` parameter—if any—is used to create the final output color. If either color profile is `nil`, QuickDraw GX uses the default color profile in its place.

When converting to an indexed color space, `GXConvertColor` uses the color set specified by the `aSet` parameter as the color set for the returned color. It returns the closest existing color in the color set.

When converting from a color space without an alpha channel to one with an alpha channel, `GXConvertColor` gives the alpha channel value maximum opacity. When converting from a color space with an alpha channel to one without an alpha channel, the alpha-channel value is lost.

When converting from a color space with colors to a luminance-based (grayscale) color space, the color information is lost but `GXConvertColor` preserves luminance (overall lightness or brightness).

When converting between color spaces with different color packings (as from `gxRGB32Space` to `gxRGB16Space` or `gxRGBSpace`), `GXConvertColor` truncates or expands individual color-component values as appropriate.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`color_is_nil`
`colorSpace_out_of_range` (debugging version)

Warnings

`colorSet_index_out_of_range`

SEE ALSO

Color spaces are described in the section “Color Spaces” beginning on page 4-6. Color matching is described in the section “Color Conversion and Color Matching” beginning on page 4-26, and in the section “Converting and Matching Colors” beginning on page 4-41.

Color Set Functions

This section describes the functions with which you create color set objects, manipulate color set object properties, and retrieve and replace colors in a color set.

Creating and Manipulating Color Set Objects

The functions in this section allow you to create and manipulate color sets as QuickDraw GX objects.

GXGetDefaultColorSet

You can use the `GXGetDefaultColorSet` function to obtain a reference to the default color set object for a given pixel depth.

```
gxColorSet GXGetDefaultColorSet(long pixelDepth);
```

`pixelDepth` The pixel size of the color set.

function result A reference to the default color set with the specified pixel depth.

DESCRIPTION

Note that the return value of this function is a reference to the actual default color set object, not a copy of it. If you edit the color set returned by this function, you alter the actual default object that the system uses when creating new color set objects.

The valid values for `pixelDepth` are 1, 2, 4, and 8. Bitmaps with other pixel depths cannot use indexed color space.

You can also alter a default color set object using the `GXSetDefaultColorSet` function, described in the next section.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`invalid_pixelSize` (debugging version)

Colors and Color-Related Objects

SEE ALSO

Default color set objects are discussed in the section “Default Color Sets” on page 4-34. To modify a default color set object, use the `GXSetDefaultColorSet` function, described next.

To create a new color set object, use the `GXNewColorSet` function, described on page 4-64.

GXSetDefaultColorSet

You can use the `GXSetDefaultColorSet` function to replace the default color set object for a particular pixel depth.

```
void GXSetDefaultColorSet(gxColorSet target, long pixelDepth);
```

`target` A reference to the color set object to make the new default.

`pixelDepth` The pixel size of the color set.

DESCRIPTION

The `GXSetDefaultColorSet` function replaces an existing default color set with the color set specified by the `target` parameter. The pixel depth of the `target` color set determines which default color set is replaced.

This function disposes of the old default color set and increments the owner count of the new default color set.

*ERRORS, WARNINGS, AND NOTICES***Errors**

<code>out_of_memory</code>	
<code>colorSet_is_nil</code>	
<code>invalid_colorSet_count</code>	(debugging version)
<code>invalid_pixelSize</code>	(debugging version)

SEE ALSO

Default color set objects are discussed in the section “Default Color Sets” on page 4-34. To obtain a copy of a default color set object, use the `GXGetDefaultColorSet` function, described in the previous section.

To create a new color set object, use the `GXNewColorSet` function, described next.

GXNewColorSet

You can use the `GXNewColorSet` function to create a new color set object.

```
gxColorSet GXNewColorSet(gxColorSpace space, long count,
                          const gxSetColor colors[]);
```

<code>space</code>	The color space of the color set. You may not specify <code>gxIndexedSpace</code> for this parameter.
<code>count</code>	The size of the color space; the number of color values it contains.
<code>colors</code>	The array of color values that make up the color set.

function result A reference to the newly created color set object.

DESCRIPTION

The `GXNewColorSet` function creates a color set object with an owner count of 1 and returns a reference to it as the function result. You specify the number of colors in the color set in the `count` parameter, and pass the colors to the function in the `colors` array. Note that the array must contain color values of type `gxSetColor`.

You do not use this function to obtain a copy of a default color set; the `colors` array must contain one or more elements. If it does not, `GXNewColorSet` posts a `color_is_nil` error. If you specify `gxIndexedSpace` for the `space` parameter, this function posts a `colorSpace_out_of_range` error.

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewColorSet` function creates a color set object; you are responsible for disposing of that object when you no longer need it.

The current implementation of QuickDraw GX restricts the number of colors in a color set to a maximum of 65,535.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>color_is_nil</code>	
<code>count_is_less_than_zero</code>	(debugging version)
<code>colorSpace_out_of_range</code>	(debugging version)
<code>number_of_colors_exceeds_implementation_limit</code>	

SEE ALSO

The `gxSetColor` union is described on page 4-56.

To obtain a copy of a default color set object, use the `GXGetDefaultColorSet` function, described on page 4-62.

GXDisposeColorSet

You can use the `GXDisposeColorSet` function to release a reference to a color set object.

```
void GXDisposeColorSet(gxColorSet target);
```

`target` A reference to the color set to dispose of.

DESCRIPTION

The `GXDisposeColorSet` function decrements the owner count of the color set specified by the `target` parameter and releases any memory used by the color set if the owner count goes to 0.

SPECIAL CONSIDERATIONS

If you attempt to dispose of a color set object used by an onscreen view device, this function posts a `colorSet_access_restricted` warning.

ERRORS, WARNINGS, AND NOTICES**Errors**

`colorSet_is_nil`

Warnings

`colorSet_access_restricted` (debugging version)

SEE ALSO

Owner counts are discussed in the section “Copying, Comparing, and Cloning Color Sets and Color Profiles” beginning on page 4-44, and in the section “Manipulating Owner Counts” beginning on page 4-46. To examine the owner count of a color set, use the `GXGetColorSetOwners` function, described on page 4-69.

GXCopyToColorSet

You can use the `GXCopyToColorSet` function to copy the contents of one existing color set to another, or to create a new color set and copy the contents of an existing color set into it.

```
gxColorSet GXCopyToColorSet(gxColorSet target, gxColorSet source);
```

`target` A reference to the color set to copy the source color set's contents into. If you specify `nil` for this parameter, the function creates a new color set.

`source` A reference to the color set whose contents you want to copy.

function result A reference to the color set copy.

DESCRIPTION

The `GXCopyToColorSet` function copies the contents of an existing color set object to another or it creates a new color set object and copies the contents of an existing color set object to it. The function copies the color space and color values and tag list (but not the owner count) of the color set object specified by the `source` parameter into the color set object specified by the `target` parameter. It clones, but does not copy, the tag objects in the tag list.

If you specify `nil` for the `target` parameter, `GXCopyToColorSet` creates a new color set object and copies the source color set's properties, including the owner count and tag list, into it.

You can use the `GXCopyToColorSet` function to create a copy of a color set and then modify it without changing the original.

SPECIAL CONSIDERATIONS

If you specify `nil` for the `target` parameter and no error occurs, the `GXCopyToColorSet` function creates a color set object; you are responsible for disposing of that object when you no longer need it.

If you specify a color set object used by an onscreen view device as the `target`, this function posts a `colorSet_access_restricted` warning.

ERRORS, WARNINGS, AND NOTICES**Errors**

out_of_memory
colorSet_is_nil

Warnings

colorSet_access_restricted (debugging version)

SEE ALSO

To create a new color set that is not a copy of an existing color set, use the `GXNewColorSet` function, described on page 4-64.

To compare two color set objects, use the `GXEqualColorSet` function, described in the next section.

GXEqualColorSet

You can use the `GXEqualColorSet` function to determine whether two color set objects are equal.

```
boolean GXEqualColorSet(gxColorSet one, gxColorSet two);
```

one A reference to one of the color sets to test for equality.

two A reference to the other color set to test for equality.

function result true if the two color sets are equal; false otherwise.

DESCRIPTION

The `GXEqualColorSet` function tests two color set objects for equality. For two color sets to be equal, they must have the same color space and identical color values—in the same order. Their owner counts and tag lists need not be identical.

ERRORS, WARNINGS, AND NOTICES**Errors**

out_of_memory
colorSet_is_nil

SEE ALSO

To make a copy of a color set object that is equal by the criteria of this function, use the `GXCopyToColorSet` function, described in the previous section.

GXCloneColorSet

You can use the `GXCloneColorSet` function to clone a color set—that is, to add a reference to it and increment its owner count.

```
gxColorSet GXCloneColorSet(gxColorSet source);
```

source A reference to the color set to clone.

function result A reference to the cloned color set.

DESCRIPTION

The `GXCloneColorSet` function increments the owner count of the color set referenced in the `source` parameter. You typically use this function when you want to create another reference to an existing color set rather than creating a distinct copy of the color set.

This function returns as its function result a reference to the color set—the same reference you pass in as the `source` parameter. It also increments the color set’s owner count.

SPECIAL CONSIDERATIONS

If you attempt to clone a color set object used by an onscreen view device, this function posts a `colorSet_access_restricted` warning.

ERRORS, WARNINGS, AND NOTICES

Errors

`colorSet_is_nil`

Warnings

`colorSet_access_restricted` (debugging version)

SEE ALSO

Owner counts for color set objects are discussed in the section “Copying, Comparing, and Cloning Color Sets and Color Profiles” beginning on page 4-44, and in the section “Manipulating Owner Counts” beginning on page 4-46.

To examine the owner count of a color set, use the `GXGetColorSetOwners` function, described on page 4-69. To decrement the owner count of a color set, use the `GXDisposeColorSet` function, described on page 4-65.

Manipulating Color Set Object Properties

The functions described in this section allow you to manipulate the common object properties of color sets: owner count and tag list. Functions for manipulating the colors in a color set are described in the section “Retrieving and Replacing Colors in a Color Set” beginning on page 4-73.

GXGetColorSetOwners

You can use the `GXGetColorSetOwners` function to determine the number of references to a particular color set object.

```
long GXGetColorSetOwners(gxColorSet source);
```

source A reference to the color set object to find the owner count of.

function result The owner count of the color set object referenced in the **source** parameter.

DESCRIPTION

The `GXGetColorSetOwners` function returns the owner count of the referenced color set. The owner count is the current number of references to the color set object.

ERRORS, WARNINGS, AND NOTICES

Errors

`colorSet_is_nil`

SEE ALSO

Owner counts for color set objects are discussed in the section “Copying, Comparing, and Cloning Color Sets and Color Profiles” beginning on page 4-44, and in the section “Manipulating Owner Counts” beginning on page 4-46.

GXGetColorSetTags

You can use the `GXGetColorSetTags` function to examine one or more of the tag objects associated with a color set object.

```
long GXGetColorSetTags(gxColorSet source, long tagType,
                      long index, long count, gxTag items[]);
```

<code>source</code>	A reference to the color set object to examine the tag list of.
<code>tagType</code>	The type of tag object to search for. A value of 0 indicates that you want to look for all tag types.
<code>index</code>	The (1-based) index of the first such tag reference to return.
<code>count</code>	The number of tag references to return
<code>items</code>	An array to hold the returned tag references.

function result The number of tag references found that fit the criteria.

DESCRIPTION

The `GXGetColorSetTags` function searches the tag list of the `source` color set object for references to tag objects with the tag type specified by the `tagType` parameter. If you specify 0 for the `tagType` parameter, the `GXGetColorSetTags` function searches all tag types.

You can use the `index` and the `count` parameters to specify which tag references of the appropriate type the `GXGetColorSetTags` function should return. The `index` parameter indicates the first tag reference to return and the `count` parameter indicates how many tag references to return. The `index` parameter must be greater than 0. The `count` parameter must be greater than 0 or equal to the `gxSelectToEnd` constant (-1), which indicates that all tag references (starting with the tag reference indicated by the `index` parameter) should be returned.

If you pass a value other than `nil` for the `items` parameter, the `GXGetColorSetTags` function returns in it the tag references that were found. Regardless of the value you pass for `items`, the function result is the number of tag references found that fit the criteria.

Colors and Color-Related Objects

ERRORS, WARNINGS, AND NOTICES

Errors

```

out_of_memory
colorSet_is_nil
index_is_less_than_one    (debugging version)
count_is_less_than_one    (debugging version)

```

Warnings

```

index_out_of_range
count_out_of_range

```

SEE ALSO

Tag objects are discussed in the chapter “Tag Objects” in this book.

To change the set of tag references associated with a color set object, use the `GXSetColorSetTags` function, described in the next section.

GXSetColorSetTags

You can use the `GXSetColorSetTags` function to add, remove, or replace tag objects associated with a color set object.

```

void GXSetColorSetTags(gxColorSet target, long tagType,
                       long index, long oldCount,
                       long newCount, const gxTag items[]);

```

<code>target</code>	A reference to the color set object to alter the tag list of.
<code>tagType</code>	The type of tag objects to replace. A value of 0 indicates that you want to replace tags of all types.
<code>index</code>	The (1-based) index of the first tag reference (to a tag object of the appropriate type) to replace.
<code>oldCount</code>	The number of tag references to replace. A value of 0 specifies that you want to insert tag references before the tag reference indicated by the <code>index</code> parameter, rather than replace tag references. A value of -1 (the <code>gxSelectToEnd</code> constant) specifies that all tag references of the requested type, starting with the tag reference indicated by the <code>index</code> parameter, should be replaced.
<code>newCount</code>	The number of tag references to insert. A value of 0 specifies that there are no tag references to insert; the existing tag references that match the criteria you specify are removed from the source color set’s tag list and disposed of.
<code>items</code>	An array of tag references to insert in the tag list.

Colors and Color-Related Objects

DESCRIPTION

The `GXSetColorSetTags` function allows you add tag references to a color set object's tag list, to remove tag references from the list, or to replace tag references in the list with new tag references. In any of these three cases, the `target` parameter specifies the color set object to be modified, the `newCount` parameter specifies the number of tag references to add, and the `items` parameter provides the new tag references.

- To add tag references, set the `oldCount` parameter to 0. Use the `tagType` and the `index` parameters to specify where to add the new tag references. (For example, if you specify `nil` for the `tagType` parameter and 1 for the `index` parameter, this function inserts the new tag references before the current tag references. If you specify a value other than `nil` for the `tagType` parameter and a value of 2 for the `index` parameter, the function inserts the new tag references before the second tag reference with a tag type matching the `tagType` parameter.)
- To remove tag references, set the `newCount` parameter to 0 and the `items` parameter to `nil`. You can use the `index` and the `oldCount` parameters to specify which tag references of the specified type should be removed. The `index` parameter indicates the first tag reference of the specified type to remove and the `oldCount` parameter indicates how many tag references of the specified type to remove.
- To replace tag references, use the `tagType`, `index`, and `oldCount` parameters to indicate which tag references to replace, and use the `newCount` and `items` parameters to specify the new tag references to add. If `newCount` is greater than `oldCount`, the extra tag references are placed immediately adjacent to the last tag reference replaced.

SPECIAL CONSIDERATIONS

If you attempt to modify the tag list of a color set object used by an onscreen view device, this function posts a `colorSet_access_restricted` warning.

ERRORS, WARNINGS, AND NOTICES**Errors**

<code>out_of_memory</code>	
<code>colorSet_is_nil</code>	
<code>tag_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>parameter_out_of_range</code>	(debugging version)
<code>index_is_less_than_zero</code>	(debugging version)
<code>cannot_dispose_locked_tag</code>	(debugging version)

Warnings

<code>index_out_of_range</code>	
<code>count_out_of_range</code>	
<code>colorSet_access_restricted</code>	(debugging version)

Notices (debugging version)

<code>tag_already_set</code>	
------------------------------	--

Colors and Color-Related Objects

SEE ALSO

Tag objects are discussed in the chapter “Tag Objects” in this book.

To examine the set of tag references associated with a color set object, use the `GXGetColorSetTags` function, described in the previous section.

Retrieving and Replacing Colors in a Color Set

The functions described in this section allow you to manipulate the colors in a color set. Functions for manipulating the other properties of color sets are described in the section “Manipulating Color Set Object Properties” beginning on page 4-69.

GXGetColorSet

You can use the `GXGetColorSet` function to retrieve the color values from a color set object.

```
long GXGetColorSet(gxColorSet source, gxColorSpace *space,
                  gxSetColor colors[]);
```

<code>source</code>	A reference to the color set object whose color values you want to retrieve.
<code>space</code>	A pointer to a color space value. On return, specifies the color space for the source color set.
<code>colors</code>	An array of <code>gxSetColor</code> color values. On return, contains the set of color values in the source color set.

function result The number of color values in the source color set.

DESCRIPTION

The `GXGetColorSet` function retrieves the color values from the source color set and returns them in the `colors` array. It also returns the color set’s color space in the location pointed to by the `space` parameter. The function result is the number of colors returned in the `colors` array.

Before calling `GXGetColorSet`, you must allocate an array of sufficient size to hold the color-value array of the color set. If instead you pass `nil` for the `colors` parameter, the function does not return any color values, but nonetheless returns (as its function result) the number of colors in the color set. Thus you can make an initial call to `GXGetColorSet` to determine the size of the array to allocate, and then call it once more to get the color values themselves.

Colors and Color-Related Objects

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
colorSet_is_nil

SEE ALSO

To replace the entire array of color values in a color set object, use the `GXSetColorSet` function, described in the next section. To retrieve some of the color values in a color set object, use the `GXGetColorSetParts` function, described on page 4-75. To replace some of the color values in a color set object, use the `GXSetColorSetParts` function, described on page 4-76.

The `gxSetColor` union is described on page 4-56.

GXSetColorSet

You can use the `GXSetColorSet` function to replace the color values of a color set object.

```
void GXSetColorSet(gxColorSet target, gxColorSpace space,
                  long count, const gxSetColor colors[]);
```

<code>target</code>	A reference to the color set object whose color values you want to replace.
<code>space</code>	The new color space for the color set referenced in the <code>target</code> parameter.
<code>count</code>	The number of color values in the <code>colors</code> array.
<code>colors</code>	The array of color values to assign to the color set.

DESCRIPTION

The `GXSetColorSet` function assigns the specified color space and color values to the target color set. If `gxNoSpace` is passed in the `space` parameter, the color space is unchanged. If the `colors` array is `nil` and if `count` is zero, the color set remains unchanged.

SPECIAL CONSIDERATIONS

If you attempt to modify the color values of a color set object used by an onscreen view device, this function posts a `colorSet_access_restricted` warning.

The current implementation of QuickDraw GX restricts the number of colors in a color set to a maximum of 65,535.

Colors and Color-Related Objects

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>colorSet_is_nil</code>	
<code>inconsistent_parameters</code>	(debugging version)
<code>count_is_less_than_zero</code>	(debugging version)
<code>colorSpace_out_of_range</code>	(debugging version)
<code>number_of_colors_exceeds_implementation_limit</code>	

Warnings

<code>colorSet_access_restricted</code>	(debugging version)
---	---------------------

SEE ALSO

To retrieve the entire array of color values from a color set object, use the `GXGetColorSet` function, described in the previous section. To retrieve some of the color values in a color set object, use the `GXGetColorSetParts` function, described in the next section. To replace some of the color values in a color set object, use the `GXSetColorSetParts` function, described on page 4-76.

The `gxSetColor` union is described on page 4-56.

GXGetColorSetParts

You can use the `GXGetColorSetParts` function to retrieve specified colors from a color set object.

```
long GXGetColorSetParts(gxColorSet source, long index, long count,
                        gxColorSpace *space, gxSetColor data[]);
```

<code>source</code>	A reference to the color set object whose color values you want to retrieve.
<code>index</code>	The first color value to retrieve. To retrieve the first color value in the color set, specify 1 for this parameter.
<code>count</code>	The number of color values to retrieve. Specify <code>gxSelectToEnd</code> to retrieve all color values in the color set including and beyond <code>index</code> .
<code>space</code>	A pointer to a color space value. On return, specifies the color space for the source color set.
<code>data</code>	An array of <code>gxSetColor</code> color values. On return, contains the specified subset of color values from the source color set.

function result The number of color values in the range specified by `index` and `count`.

Colors and Color-Related Objects

DESCRIPTION

The `GXGetColorSetParts` function retrieves the specified color values from the source color set and returns them in the data array. It also returns the color set's color space in the location pointed to by the `space` parameter. The function result is the number of color values copied into the data array.

Before calling `GXGetColorSetParts`, you must allocate an array of sufficient size to hold the specified number of color values. If instead you pass `nil` for the `data` parameter, the function does not return any color values, but nonetheless returns (as its function result) the number of colors in the specified range. Thus you can make an initial call to `GXGetColorSetParts` to determine the size of the array to allocate, and then call it once more to get the color values themselves.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`colorSet_is_nil`
`index_is_less_than_one` (debugging version)
`count_is_less_than_one` (debugging version)

Warnings

`index_out_of_range`
`count_out_of_range`

SEE ALSO

To retrieve the entire array of color values from a color set object, use the `GXGetColorSet` function, described on page 4-73. To replace the entire array of color values in a color set object, use the `GXSetColorSet` function, described in the previous section. To replace some of the color values in a color set object, use the `GXSetColorSetParts` function, described in the next section.

The `gxSetColor` union is described on page 4-56.

GXSetColorSetParts

You can use the `GXSetColorSetParts` function to replace specified colors in a color set object.

```
void GXSetColorSetParts(gxColorSet target, long index,
                        long oldCount, long newCount,
                        const gxSetColor data[]);
```

<code>target</code>	A reference to the color set object whose color values you want to modify.
<code>index</code>	The first color value to replace. To replace the first color value in the color set, specify 1 for this parameter.

Colors and Color-Related Objects

<code>oldCount</code>	The number of color values to replace. Specify <code>gxSelectToEnd</code> to replace all color values in the color set including and beyond <code>index</code> .
<code>newCount</code>	The number of new color values to add; that is, the number of color values in the data array.
<code>data</code>	The array of color values to add to the color set.

DESCRIPTION

The `GXSetColorSetParts` function assigns the specified color values to the target color set, starting at the location specified by `index` after first removing the number of existing color values specified by `oldCount`.

This function does not accept the `gxSetToNil` constant for the `data` parameter. If you want to simply remove colors, pass 0 for `newCount`.

The current implementation of QuickDraw GX restricts the number of colors in a color set to a maximum of 65,535.

SPECIAL CONSIDERATIONS

If you attempt to modify the color values of a color set object used by an onscreen view device, this function posts a `colorSet_access_restricted` warning.

ERRORS, WARNINGS, AND NOTICES**Errors**

<code>out_of_memory</code>	
<code>colorSet_is_nil</code>	
<code>inconsistent_parameters</code>	(debugging version)
<code>index_is_less_than_zero</code>	(debugging version)
<code>count_is_less_than_zero</code>	(debugging version)
<code>number_of_colors_exceeds_implementation_limit</code>	

Warnings

<code>index_out_of_range</code>	
<code>count_out_of_range</code>	
<code>colorSet_access_restricted</code>	(debugging version)

SEE ALSO

To retrieve the entire array of color values from a color set object, use the `GXGetColorSet` function, described on page 4-73. To replace the entire array of color values in a color set object, use the `GXSetColorSet` function, described on page 4-74. To retrieve some of the color values in a color set object, use the `GXSetColorSetParts` function, described in the previous section.

The `gxSetColor` union is described on page 4-56.

Color Profile Functions

This section describes the functions with which you create color profile objects, manipulate color profile object properties, and retrieve and replace profile information.

Creating and Manipulating Color Profile Objects

The functions in this section allow you to create and manipulate color profiles as QuickDraw GX objects. For descriptions of functions that manipulate the properties of color profile objects, see the sections “Manipulating Color Profile Object Properties” beginning on page 4-84 and “Retrieving and Replacing Profile Information” beginning on page 4-88.

GXGetDefaultColorProfile

You can use the `GXGetDefaultColorProfile` function to obtain a reference to the default color profile object.

```
gxColorProfile GXGetDefaultColorProfile(void);
```

function result A reference to the default color profile.

DESCRIPTION

The default color profile is the color profile for the Apple 13-inch color monitor. When converting or matching colors, QuickDraw GX assumes the default color profile for any color, bitmap, or transfer mode whose color profile property is `nil`.

Note that the return value of this function is a reference to the actual default color profile object, not a copy of it. You should not make changes to the profile; if you edit it (for example, by calling `GXLockProfile` and `GXGetProfileStructure`), you alter the actual default profile that QuickDraw GX uses when creating new color profile objects.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

Colors and Color-Related Objects

SEE ALSO

The default color profile object is discussed in the section “The Default Color Profile” beginning on page 4-37.

To create a copy of the default color profile object, you can use the `GXCopyToColorProfile` function, described on page 4-81.

To create a new color profile, use the `GXNewColorProfile` function, described next.

GXNewColorProfile

You can use the `GXNewColorProfile` function to create a new color profile object.

```
gxColorProfile GXNewColorProfile(long size,
                                void *colorProfileData);
```

size The size in bytes of the profile data to assign to the new color profile object.

colorProfileData A pointer to the profile data to assign to the new color profile object.

function result A reference to the newly created color profile object.

DESCRIPTION

The `GXNewColorProfile` function creates a color profile object with an owner count of 1 from the profile data that you supply. The new color profile object is not a copy of the default color profile.

If you specify a nonzero value for the `size` parameter, you must pass a `ColorSync` color profile structure to `GXNewColorProfile`. The function does not check for the validity of the profile data, but if the `colorProfileData` parameter is `nil` and the `size` parameter is nonzero the function posts an error.

You can create a zero-length profile by passing 0 for the `size` parameter when calling this function. The effect of a zero-length profile is to inhibit color matching.

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewColorProfile` function creates a color profile object; you are responsible for disposing of that object when you no longer need it.

Colors and Color-Related Objects

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
parameter_is_nil (debugging version)

SEE ALSO

Zero-length profiles are described in the section “Zero-Length Profiles” on page 4-37.

The format of the profile data in a color profile object is described in the section “Profile Data” beginning on page 4-36. The ColorSync Utilities are described in *Inside Macintosh: Advanced Color Imaging*.

To obtain a reference to the default color profile, use the `GXGetDefaultColorProfile` function, described in the previous section.

GXDisposeColorProfile

You can use the `GXDisposeColorProfile` function to release a reference to a color profile object.

```
void GXDisposeColorProfile(gxColorProfile target);
```

target A reference to the color profile to dispose of.

DESCRIPTION

The `GXDisposeColorProfile` function decrements the owner count of the color profile object referenced in the `target` parameter and releases any memory used by the color profile if the owner count goes to 0.

SPECIAL CONSIDERATIONS

If you attempt to dispose of a color profile object used by an onscreen view device, this function posts a `colorProfile_access_restricted` warning.

ERRORS, WARNINGS, AND NOTICES

Errors

colorProfile_is_nil

Warnings

colorProfile_access_restricted (debugging version)

SEE ALSO

Owner counts are discussed in the section “Copying, Comparing, and Cloning Color Sets and Color Profiles” beginning on page 4-44, and in the section “Manipulating Owner Counts” beginning on page 4-46. To examine the owner count of a color profile, use the `GXGetColorProfileOwners` function, described on page 4-84.

GXCopyToColorProfile

You can use the `GXCopyToColorProfile` function to copy the contents of an existing color profile object into another or to create a new color profile object and copy the contents of an existing color profile into it.

```
gxColorProfile GXCopyToColorProfile(gxColorProfile target,
                                     gxColorProfile source);
```

target A reference to the color profile to copy the source contents into. If you specify `nil` for this parameter, the `GXCopyToColorProfile` function creates a new color profile.

source A reference to the color profile object whose contents you want to copy.

function result A reference to the color profile copy.

DESCRIPTION

The `GXCopyToColorProfile` function either copies the contents of an existing color profile object to another or creates a new color profile object and copies the contents of an existing color profile object to it. The function copies the profile data and tag list (but not the owner count) of the color profile specified by the `source` parameter into the color profile specified by the `target` parameter. It clones, but does not copy, the tag objects in the tag list.

If you specify `nil` for the `target` parameter, `GXCopyToColorProfile` creates a new color profile object and copies the source properties, including the owner count and tag list, into it.

You can use the `GXCopyToColorProfile` function to create a copy of a color profile and then modify it without changing the original.

SPECIAL CONSIDERATIONS

If you specify `nil` for the `target` parameter and no error occurs, the `GXCopyToColorProfile` function creates a color profile object; you are responsible for disposing of that object when you no longer need it.

If you specify a color profile object used by an onscreen view device as the `target`, this function posts a `colorProfile_access_restricted` warning.

Colors and Color-Related Objects

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
colorProfile_is_nil

Warnings

colorProfile_access_restricted (debugging version)

SEE ALSO

To create a new color profile that is not a copy of an existing color profile, use the `GXNewColorProfile` function, described on page 4-79.

To compare two color profile objects, use the `GXEqualColorProfile` function, described next.

GXEqualColorProfile

You can use the `GXEqualColorProfile` function to determine whether two color profile objects are equal.

```
boolean GXEqualColorProfile(gxColorProfile one,
                           gxColorProfile two);
```

one A reference to one of the color profiles to test for equality.

two A reference to the other color profile to test for equality.

function result true if the color profiles are equal; false otherwise.

DESCRIPTION

The `GXEqualColorProfile` function tests two color profile objects for equality. For two color profiles to be equal, they must have exactly the same profile data, although their owner counts and tag lists need not be identical.

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
colorProfile_is_nil

SEE ALSO

To make a copy of a color profile object that is equal by the criteria of this function, use the `GXCopyToColorProfile` function, described in the previous section.

GXCloneColorProfile

You can use the `GXCloneColorProfile` function to clone a color profile—that is, to add a reference to it and increment its owner count.

```
gxColorProfile GXCloneColorProfile(gxColorProfile source);
```

source A reference to the color profile to clone.

function result A reference to the cloned color profile.

DESCRIPTION

The `GXCloneColorProfile` function increments the owner count of the color profile referenced in the `source` parameter. You typically use this function when you want to create another reference to an existing color profile rather than creating a distinct copy of the color profile.

This function returns as its function result a reference to the color profile—the same reference you pass in as the `source` parameter. It also increments the color profile's owner count.

SPECIAL CONSIDERATIONS

If you attempt to clone a color profile object used by an onscreen view device, this function posts a `colorProfile_access_restricted` warning.

ERRORS, WARNINGS, AND NOTICES

Errors

`colorProfile_is_nil`

Warnings

`colorProfile_access_restricted` (debugging version)

SEE ALSO

Owner counts for color profile objects are discussed in the section “Copying, Comparing, and Cloning Color Sets and Color Profiles” beginning on page 4-44, and in the section “Manipulating Owner Counts” beginning on page 4-46.

To examine the owner count of a color profile, use the `GXGetColorProfileOwners` function, described on page 4-84. To decrement the owner count of a color profile, use the `GXDisposeColorProfile` function, described on page 4-80.

Manipulating Color Profile Object Properties

The functions described in this section allow you to manipulate the common object properties of color profile objects: owner count and tag list. For descriptions of functions that manipulate the profile data of color profile objects, see the section “Retrieving and Replacing Profile Information” beginning on page 4-88. For descriptions of functions that allow you to create and manipulate color profiles as QuickDraw GX objects, see the section “Creating and Manipulating Color Profile Objects” beginning on page 4-78.

GXGetColorProfileOwners

You can use the `GXGetColorProfileOwners` function to determine the number of references to a particular color profile object.

```
long GXGetColorProfileOwners(gxColorProfile source);
```

`source` A reference to the color profile object to find the owner count of.

function result The owner count of the source color profile object.

DESCRIPTION

The `GXGetColorProfileOwners` function returns the owner count of the referenced color profile object. The owner count is the current number of references to the color profile object.

ERRORS, WARNINGS, AND NOTICES

Errors

`colorProfile_is_nil`

SEE ALSO

Owner counts for color profile objects are discussed in the section “Copying, Comparing, and Cloning Color Sets and Color Profiles” beginning on page 4-44, and in the section “Manipulating Owner Counts” beginning on page 4-46.

GXGetColorProfileTags

You can use the `GXGetColorProfileTags` function to examine one or more of the tag objects associated with a color profile object.

```
long GXGetColorProfileTags(gxColorProfile source, long tagType,
                           long index, long count, gxTag items[]);
```

<code>source</code>	A reference to the color profile object to examine the tag list of.
<code>tagType</code>	The type of tag object to search for. A value of 0 indicates that you want to look for all tag types.
<code>index</code>	The (1-based) index of the first such tag reference to return.
<code>count</code>	The number of tag references to return.
<code>items</code>	An array to hold the returned tag references.

function result The number of tag references found that fit the criteria.

DESCRIPTION

The `GXGetColorProfileTags` function searches the tag list of the `source` color profile object for references to tag objects with the tag type specified by the `tagType` parameter. If you specify 0 for the `tagType` parameter, the `GXGetColorProfileTags` function searches all tag types.

You can use the `index` and the `count` parameters to specify which tag references of the appropriate type the `GXGetColorProfileTags` function should return. The `index` parameter indicates the first tag reference to return and the `count` parameter indicates how many tag references to return. The `index` parameter must be greater than 0. The `count` parameter must be greater than 0 or equal to the `gxSelectToEnd` constant (-1), which indicates that all tag references (starting with the tag reference indicated by the `index` parameter) should be returned.

If you pass a value other than `nil` for the `items` parameter, the `GXGetColorProfileTags` function returns in it the tag references that were found. Regardless of the value you pass for `items`, the function result is the number of tag references found that fit the criteria.

ERRORS, WARNINGS, AND NOTICES

Errors

```
out_of_memory
colorProfile_is_nil
index_is_less_than_one    (debugging version)
count_is_less_than_one    (debugging version)
```

Warnings

```
index_out_of_range
count_out_of_range
```

Colors and Color-Related Objects

SEE ALSO

Tag objects are discussed in the chapter “Tag Objects” in this book.

To change the set of tag references associated with a color profile object, use the `GXSetColorProfileTags` function, described in the next section.

GXSetColorProfileTags

You can use the `GXSetColorProfileTags` function to add, remove, or replace tag objects associated with a color profile object.

```
void GXSetColorProfileTags(gxColorProfile target, long tagType,
                           long index, long oldCount,
                           long newCount, const gxTag items[]);
```

<code>target</code>	A reference to the color profile object to alter the tag list of.
<code>tagType</code>	The type of tag objects to replace. A value of 0 indicates that you want to replace tags of all types.
<code>index</code>	The (1-based) index of the first tag reference (to a tag object of the appropriate type) to replace.
<code>oldCount</code>	The number of tag references to replace. A value of 0 specifies that you want to insert tag references before the tag reference indicated by the <code>index</code> parameter, rather than replace tag references. A value of -1 (the <code>gxSelectToEnd</code> constant) specifies that all tag references of the requested type, starting with the tag reference indicated by the <code>index</code> parameter, should be replaced.
<code>newCount</code>	The number of tag references to insert. A value of 0 specifies that there are no tag references to insert; the existing tag references that match the criteria you specify are removed from the source color profile’s tag list and disposed of.
<code>items</code>	An array of tag references to insert in the tag list.

DESCRIPTION

The `GXSetColorProfileTags` function allows you add tag references to a color profile object’s tag list, to remove tag references from the list, or to replace tag references in the list with new tag references. In any of these three cases, the `target` parameter specifies the color profile object to be modified, the `newCount` parameter specifies the number of tag references to add, and the `items` parameter provides the new tag references.

Colors and Color-Related Objects

- To add tag references, set the `oldCount` parameter to 0. Use the `tagType` and the `index` parameters to specify where to add the new tag references. (For example, if you specify `nil` for the `tagType` parameter and 1 for the `index` parameter, this function inserts the new tag references before the current tag references. If you specify a value other than `nil` for the `tagType` parameter and a value of 2 for the `index` parameter, the function inserts the new tag references before the second tag reference with a tag type matching the `tagType` parameter.)
- To remove tag references, set the `newCount` parameter to 0 and the `items` parameter to `nil`. You can use the `index` and the `oldCount` parameters to specify which tag references of the specified type should be removed. The `index` parameter indicates the first tag reference of the specified type to remove and the `oldCount` parameter indicates how many tag references of the specified type to remove.
- To replace tag references, use the `tagType`, `index`, and `oldCount` parameters to indicate which tag references to replace, and use the `newCount` and `items` parameters to specify the new tag references to add. If `newCount` is greater than `oldCount`, the extra tag references are placed immediately adjacent to the last tag reference replaced.

SPECIAL CONSIDERATIONS

If you attempt to modify the tag list of a color profile object used by an onscreen view device, this function posts a `colorProfile_access_restricted` warning.

ERRORS, WARNINGS, AND NOTICES**Errors**

<code>out_of_memory</code>	
<code>colorSet_is_nil</code>	
<code>tag_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>parameter_out_of_range</code>	(debugging version)
<code>index_is_less_than_zero</code>	(debugging version)
<code>cannot_dispose_locked_tag</code>	(debugging version)

Warnings

<code>index_out_of_range</code>	
<code>count_out_of_range</code>	
<code>colorProfile_access_restricted</code>	(debugging version)

Notices (debugging version)

<code>tag_already_set</code>

SEE ALSO

Tag objects are discussed in the chapter “Tag Objects” in this book.

To examine the set of tag references associated with a color profile object, use the `GXGetColorProfileTags` function, described in the previous section.

Retrieving and Replacing Profile Information

The functions described in this section allow you to manipulate the profile data of color profile objects. For descriptions of functions that manipulate the common object properties of color profile object, see the section “Manipulating Color Profile Object Properties” beginning on page 4-84. For descriptions of functions that allow you to create and manipulate color profiles as QuickDraw GX objects, see the section “Creating and Manipulating Color Profile Objects” beginning on page 4-78.

GXGetColorProfile

You can use the `GXGetColorProfile` function to retrieve the profile data from a color profile object.

```
long GXGetColorProfile(gxColorProfile source,
                      void *colorProfileData);
```

`source` A reference to the color profile object to get the profile data from.

`colorProfileData` A pointer to a buffer. On return, the buffer contains the profile data for the source color profile.

function result The size in bytes of the source color profile’s profile data.

DESCRIPTION

The `GXGetColorProfile` function returns the profile data from the source color profile in the buffer pointed to by the `responses` parameter. It also returns the size of the profile data as the function result.

The profile data returned by this function is a ColorSync color profile structure (type `CMProfile`).

If you specify `nil` for the `colorProfileData` parameter, this function does not return the profile data, but it nevertheless returns a correct value for the size of the profile response structure in the function result. Thus you can make an initial call to `GXGetColorProfile` to determine the size of buffer to allocate, and then call it once more to get the profile data itself.

Colors and Color-Related Objects

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
colorProfile_is_nil

SEE ALSO

To replace the profile data in a color profile object, use the `GXSetColorProfile` function, described in the next section.

The format of the profile data in a color profile object is described in the section “Profile Data” beginning on page 4-36. The ColorSync Utilities, including the `CMProfile` data type, are described in *Inside Macintosh: Advanced Color Imaging*.

GXSetColorProfile

You can use the `GXSetColorProfile` function to assign profile data to a color profile object.

```
void GXSetColorProfile(gxColorProfile target, long size,
                      void *colorProfileData);
```

<code>target</code>	A reference to the color profile object whose profile data you want to change.
<code>size</code>	The size in bytes of the profile data to assign to the target color profile.
<code>colorProfileData</code>	A pointer to the profile data.

DESCRIPTION

The `GXSetColorProfile` function assigns the specified profile data to the target color profile. If you specify a nonzero value for the `size` parameter, the pointer to the profile data must not be `nil`. It should be in the form of a valid ColorSync color profile structure (type `CMProfile`), although the function does not actually verify this.

If you pass 0 for the `size` parameter to this function, QuickDraw GX converts this profile into a zero-length profile, which you can use to inhibit color matching.

SPECIAL CONSIDERATIONS

If you attempt to alter the profile data of a color profile object used by an onscreen view device, this function posts a `colorProfile_access_restricted` warning.

Colors and Color-Related Objects

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>colorProfile_is_nil</code>	
<code>inconsistent_parameters</code>	(debugging version)
<code>parameter_out_of_range</code>	(debugging version)

Warnings

<code>colorProfile_access_restricted</code>	(debugging version)
---	---------------------

SEE ALSO

To examine the profile data in a color profile object, use the `GXGetColorProfile` function, described in the previous section.

Zero-length profiles are described in the section “Zero-Length Profiles” on page 4-37.

The format of the profile data in a color profile object is described in the section “Profile Data” beginning on page 4-36. The ColorSync Utilities, including the `CMPProfile` data type, are described in *Inside Macintosh: Advanced Color Imaging*.

GXLockColorProfile

You can use the `GXLockColorProfile` function to load a color profile object into memory and lock its profile data into a fixed memory location.

```
void GXLockColorProfile (gxColorProfile source);
```

`source` A reference to the color profile to be loaded and locked.

DESCRIPTION

The `GXLockColorProfile` function prevents a color profile from being relocated, so that you can manipulate its profile data directly in QuickDraw GX memory rather than working with a copy of it in application memory.

To directly edit the color profile, call `GXLockColorProfile` followed by `GXGetColorProfileStructure`; after editing, call `GXUnlockColorProfile`.

SPECIAL CONSIDERATIONS

To avoid fragmenting the QuickDraw GX heap, call the `GXUnlockColorProfile` function as soon as possible after calling `GXLockColorProfile`.

In low memory situations with a fragmented heap, QuickDraw GX can unlock locked objects without warning. Be careful about making memory-intensive calls when you are working with a locked color profile.

Colors and Color-Related Objects

*ERRORS, WARNINGS, AND NOTICES***Errors**

out_of_memory
colorProfile_is_nil

SEE ALSO

The GXUnlockColorProfile and GXGetColorProfileStructure functions are described in the next two sections.

GXUnlockColorProfile

You can use the GXUnlockColorProfile function to allow QuickDraw GX to relocate, compress, or unload a color profile object that has been locked.

```
void GXUnlockColorProfile (gxColorProfile source);
```

gxColorProfile

A reference to the color profile to be unlocked.

DESCRIPTION

To directly edit the color profile, call GXLockColorProfile followed by GXGetColorProfileStructure; after editing, call GXUnlockColorProfile.

Once you call GXUnlockColorProfile, the profile data may be relocated and a pointer returned by GXGetColorProfileStructure may no longer be valid.

SPECIAL CONSIDERATIONS

To avoid fragmenting the QuickDraw GX heap, call the GXUnlockColorProfile function as soon as possible after calling GXLockColorProfile.

*ERRORS, WARNINGS, AND NOTICES***Errors**

colorProfile_is_nil

SEE ALSO

The GXLockColorProfile function is described in the previous section. The GXGetColorProfileStructure function is described next.

GXGetColorProfileStructure

You can use the `GXGetColorProfileStructure` function to get a pointer to the profile data of a color profile object.

```
void *GXGetColorProfileStructure(gxColorProfile source,
                                long *length);
```

<code>source</code>	A reference to the color profile object whose profile data you need access to.
<code>length</code>	A pointer to a long value. On return, the value specifies the size in bytes of the profile data.

function result A pointer to the profile data of the source color profile.

DESCRIPTION

The `GXGetColorProfileStructure` function determines the size of the profile data in a color profile object and returns a pointer to the data in the QuickDraw GX heap. You can use the pointer to examine or change the profile data without copying the data into your application's heap and back again.

Before calling this function, call `GXLockColorProfile` to lock the profile data in memory; after editing the profile data, call `GXUnlockColorProfile` to free the profile data for relocation.

The profile data returned by this function is a ColorSync color profile structure (type `CMProfile`).

This function is useful even if you do not intend to edit a color profile. You can use it to simply read a specific piece of color profile data, such as the white point, without having to obtain a copy of the entire profile.

SPECIAL CONSIDERATIONS

To avoid fragmenting the QuickDraw GX heap, call the `GXUnlockColorProfile` function as soon as possible after manipulating the profile data.

You cannot change the size of the profile data you access with this call. If your manipulations require a change in the size of the data, you must use `GXGetColorProfile` and `GXSetColorProfile`.

This function is rarely needed. In most situations you do not need to alter the profile data of a color profile, and when you do you can use the functions `GXGetColorProfile` and `GXSetColorProfile` to make the needed changes.

Colors and Color-Related Objects

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
colorProfile_is_nil

SEE ALSO

The `GXLockColorProfile` and `GXUnlockColorProfile` functions are described in the previous two sections.

The format of the profile data in a color profile object is described in the section “Profile Data” beginning on page 4-36. The ColorSync Utilities, including the `CMProfile` data type, are described in *Inside Macintosh: Advanced Color Imaging*.

To edit a copy of a color profile object’s profile data, rather than directly changing the data in QuickDraw GX memory, use the `GXGetColorProfile` function, described on page 4-88; to assign the edited data back to the profile, use the `GXSetColorProfile` function, described on page 4-89.

Summary of Colors and Color-Related Objects

Constants and Data Types

Color-Component Values

```
typedef unsigned short gxColorValue;
```

Color Values

```
struct gxCMYKColor{
    gxColorValue    cyan;
    gxColorValue    magenta;
    gxColorValue    yellow;
    gxColorValue    black;
};

struct gxRGBColor{
    gxColorValue    red;
    gxColorValue    green;
    gxColorValue    blue;
};

struct gxRGBAColor{
    gxColorValue    red;
    gxColorValue    green;
    gxColorValue    blue;
    gxColorValue    alpha;
};

struct gxHSVColor{
    gxColorValue    hue;
    gxColorValue    saturation;
    gxColorValue    value;
};

struct gxHLSColor{
    gxColorValue    hue;
    gxColorValue    lightness;
    gxColorValue    saturation;
};
```

Colors and Color-Related Objects

```

struct gxXYZColor {
    gxColorValue  x;
    gxColorValue  y;
    gxColorValue  z;
};

struct gxYXYColor {
    gxColorValue  capY;
    gxColorValue  x;
    gxColorValue  y;
};

struct gxLUVColor {
    gxColorValue  l;
    gxColorValue  u;
    gxColorValue  v;
};

struct gxLABColor {
    gxColorValue  l;
    gxColorValue  a;
    gxColorValue  b;
};

struct gxYIQColor{
    gxColorValue  y;
    gxColorValue  i;
    gxColorValue  q;
};

struct gxGrayAColor{
    gxColorValue  gray;
    gxColorValue  alpha;
};

typedef long gxColorIndex;

struct gxIndexedColor{
    gxColorIndex  index;
    gxColorSet    set;
};

```

Colors and Color-Related Objects

The Color Structure

```

struct gxColor{
    gxColorSpace      space;
    gxColorProfile     profile;
    union {
        struct gxCMYKColor      cmyk;
        struct gxRGBColor       rgb;
        struct gxRGBAColor      rgba;
        struct gxHSVColor       hsv;
        struct gxHLSColor       hls;
        struct gxXYZColor       xyz;
        struct gxYXYColor       yxy;
        struct gxLUVColor       luv;
        struct gxLABColor       lab;
        struct gxYIQColor       yiq;
        gxColorValue            gray;
        struct gxGrayAColor     graya;
        unsigned short          pixel16;
        unsigned long           pixel32;
        struct gxIndexedColor    indexed;
        gxColorValue            component[4];
    } element;
};

```

Color Packing

```

typedef enum {
    gxNoColorPacking      = 0x0000,    /* 16 bits/channel */
    gxAlphaSpace          = 0x0080,    /* space includes alpha channel */
    gxWord5ColorPacking   = 0x0500,    /* 5 bits/channel, right-justified */
    gxLong8ColorPacking   = 0x0800,    /* 8 bits/channel, right-justified */
    gxLong10ColorPacking  = 0x0a00,    /* 10 bits/channel, right-justified */
    gxAlphaFirstPacking   = 0x1000     /* alpha channel = 1st field in space */
} gxColorPackingTypes;

```

Color Spaces

```

enum gxColorSpaces{
    gxNoSpace      = 0,
    gxRGBSpace,
    gxCMYKSpace,
    gxHSVSpace,
    gxHLSpace,

```

Colors and Color-Related Objects

```

gxYXYSpace,
gxXYZSpace,
gxLUVSpace,
gxLABSpace,
gxYIQSpace,
gxNTSCSpace    = gxYIQSpace,
gxPALSpace      = gxYIQSpace,
gxGraySpace,
gxIndexedSpace,
gxRGBASpace     = gxRGBSpace + gxAlphaSpace,
gxGrayASpace    = gxGraySpace + gxAlphaSpace,
gxRGB16Space    = gxWord5ColorPacking + gxRGBSpace,
gxRGB32Space    = gxLong8ColorPacking + gxRGBSpace,
gxARGB32Space   = gxLong8ColorPacking + gxAlphaFirstPacking
                + gxRGBASpace,
gxCMYK32Space   = gxLong8ColorPacking + gxCMYKSpace,
gxHSV32Space    = gxLong10ColorPacking + gxHSVSpace,
gxHLS32Space    = gxLong10ColorPacking + gxHLSSpace,
gxYXY32Space    = gxLong10ColorPacking + gxYXYSpace,
gxXYZ32Space    = gxLong10ColorPacking + gxXYZSpace,
gxLUV32Space    = gxLong10ColorPacking + gxLUVSpace,
gxLAB32Space    = gxLong10ColorPacking + gxLABSpace,
gxYIQ32Space    = gxLong10ColorPacking + gxYIQSpace,
gxNTSC32Space   = gxYIQ32Space,
gxPAL32Space    = gxYIQ32Space,
};

typedef long gxColorSpace;

```

The Color Set Object

```
typedef struct gxPrivateColorSetRecord *gxColorSet;
```

The gxSetColor Union

```

union gxSetColor{
    gxCMYKColor    cmyk;
    gxRGBColor     rgb;
    gxRGBAColor    rgba;
    gxHSVColor     hsv;
    gxHLSColor     hls;
    gxXYZColor     xyz;
    gxYXYColor     yxy;
    gxLUVColor     luv;
}

```

Colors and Color-Related Objects

```

gxLABColor      lab;
gxYIQColor      yiq;
gxColorValue    gray;
gxGrayAColor    graya;
unsigned short  pixel16;
unsigned long    pixel32;
gxColorValue    component[4];
};

```

The Color Profile Object

```
typedef struct gxPrivateProfileRecord *gxColorProfile;
```

Color Functions

```

boolean GXCheckColor      (const gxColor *source, gxColorSpace space,
                           gxColorSet aSet, gxColorProfile profile);
Fixed GXGetColorDistance  (const gxColor *target, const gxColor *source);
gxColor *GXCombineColor   (gxColor *target, gxInk operand);
gxColor *GXConvertColor   (gxColor *target, gxColorSpace space,
                           gxColorSet aSet, gxColorProfile profile);

```

Color Set Functions

Creating and Manipulating Color Set Objects

```

gxColorSet GXGetDefaultColorSet
                           (long pixelDepth);
void GXSetDefaultColorSet  (gxColorSet target, long pixelDepth);
gxColorSet GXNewColorSet   (gxColorSpace space, long count,
                           const gxSetColor colors[]);
void GXDisposeColorSet     (gxColorSet target);
gxColorSet GXCopyToColorSet (gxColorSet target, gxColorSet source);
boolean GXEqualColorSet     (gxColorSet one, gxColorSet two);
gxColorSet GXCloneColorSet  (gxColorSet source);

```

Manipulating Color Set Object Properties

```

long GXGetColorSetOwners   (gxColorSet source);
long GXGetColorSetTags     (gxColorSet source, long tagType, long index,
                           long count, gxTag items[]);

```

Colors and Color-Related Objects

```
void GXSetColorSetTags      (gxColorSet target, long tagType, long index,
                             long oldCount, long newCount,
                             const gxTag items[]);
```

Retrieving and Replacing Colors in a Color Set

```
long GXGetColorSet          (gxColorSet source, gxColorSpace *space,
                             gxSetColor colors[]);

void GXSetColorSet         (gxColorSet target, gxColorSpace space,
                             long count, const gxSetColor colors[]);

long GXGetColorSetParts     (gxColorSet source, long index, long count,
                             gxColorSpace *space, gxSetColor data[]);

void GXSetColorSetParts    (gxColorSet target, long index, long oldCount,
                             long newCount, const gxSetColor data[]);
```

Color Profile Functions

Creating and Manipulating Color Profile Objects

```
gxColorProfile GXGetDefaultColorProfile
    (void);

gxColorProfile GXNewColorProfile
    (const gxProfileRecord *profile,
     const gxProfileResponse *responses);

void GXDisposeColorProfile (gxColorProfile target);

gxColorProfile GXCopyToColorProfile
    (gxColorProfile target, gxColorProfile source);

boolean GXEqualColorProfile (gxColorProfile one, gxColorProfile two);

gxColorProfile GXCloneColorProfile
    (gxColorProfile source);
```

Manipulating Color Profile Object Properties

```
long GXGetColorProfileOwners (gxColorProfile source);

long GXGetColorProfileTags   (gxColorProfile source, long tagType,
                             long index, long count, gxTag items[]);

void GXSetColorProfileTags   (gxColorProfile target, long tagType,
                             long index, long oldCount, long newCount,
                             const gxTag items[]);
```

Colors and Color-Related Objects

Retrieving and Replacing Profile Information

```
long GXGetColorProfile      (gxColorProfile source,  
                             gxProfileRecord *profile,  
                             gxProfileResponse *responses);  
  
void GXSetColorProfile      (gxColorProfile target,  
                             const gxProfileRecord *profile,  
                             const gxProfileResponse *responses);  
  
void GXLockColorProfile     (gxColorProfile source);  
void GXUnlockColorProfile   (gxColorProfile source);  
void *GXGetColorProfileStructure  
                             (gxColorProfile source, long *length);
```