

Ink Objects

This chapter describes ink objects and the functions you can use to manipulate them. Read this chapter if you create or use any kind of ink object for the QuickDraw GX shapes you create. Read this chapter also if you want to understand how QuickDraw GX uses transfer modes in drawing shapes.

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 shapes, as discussed in the chapter “Shape Objects” in this book.

Although colors are contained in ink objects, they are not discussed here. Colors are discussed in the chapter “Colors and Color-Related Objects” in this book. Other than that chapter, this chapter constitutes the complete discussion of ink objects for QuickDraw GX. Unlike for shape objects and style objects, there is no separate discussion in other books of any specific graphic or typographic uses for inks.

This chapter introduces QuickDraw GX ink objects and describes their properties. It also describes how transfer modes work in QuickDraw GX. It then shows how to use the QuickDraw GX ink-manipulation functions to

- create and manipulate ink objects
- manipulate ink object properties
- get and set an ink object’s color
- work with transfer modes

About Ink Objects

An ink object exists to provide color information about a shape. Each QuickDraw GX shape consists of a shape object, a style object, an ink object, and a transform object; the ink object associated with a shape defines the color with which the shape is drawn, as well as the transfer mode used to draw it.

QuickDraw GX identifies an individual ink object through an ink *reference*. To obtain information about an ink 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 ink object simply by comparing them for equality, and you can examine a reference to see if it is `nil`).

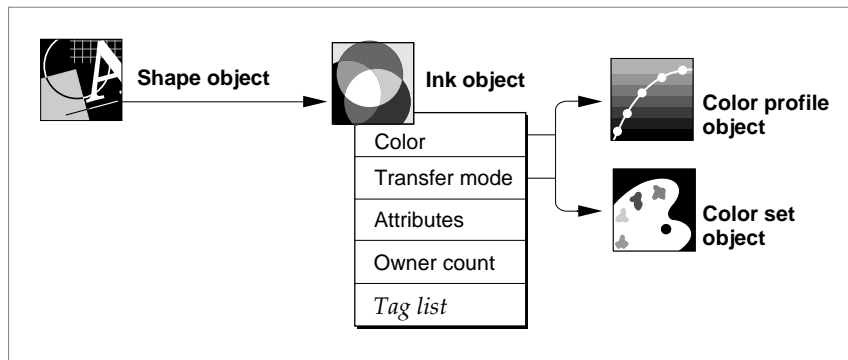
Inks are device independent. Their information is not affected by the properties of the display device to which the shapes they modify are drawn. When it draws a shape on a device, QuickDraw GX approximates as closely as possible the color specified by the shape’s ink. Device-specific color characteristics are accounted for by attaching color profiles to ink objects and by using a device’s color profile when drawing; see the chapter “Colors and Color-Related Objects” in this book for more information.

Ink Properties

The interface to ink objects is entirely procedural. You manipulate the information in an ink object by modifying its properties using QuickDraw GX functions.

Ink objects have five accessible properties, as shown in Figure 5-1. Note that, because an ink is an object and not a data structure, the order of the properties as shown in Figure 5-1 is completely arbitrary. Properties in *italics* are references to other objects.

Figure 5-1 The ink object and its properties



These are the five accessible properties in an ink object:

- **Color.** A data structure that specifies the color to use for drawing the shape associated with this ink object. Besides the numeric value of the color itself, the color structure includes a specification of the color space the color is defined in terms of, as well as optional references to two other QuickDraw GX objects, a color set and a color profile.
- **Transfer mode.** The way, or mode, of transferring the color to its destination (the screen or printed page or other location into which the shape associated with this ink is drawn). Transfer mode is a specification (such as “copy” or “XOR” or “blend”) of the interaction between the color in this ink object and the existing color or colors of the destination. With transfer mode you can make a shape opaque or transparent, draw only part of it, change its color, or combine its color with the destination color in many different ways.
The transfer mode also includes a specification of a color space and may include references to a color set and a color profile.
- **Attributes.** A set of flags that allow you to control certain properties of view ports that affect how colors appear when a shape is drawn.
- **Owner count.** The number of existing references to this ink object.
- **Tag list.** A list of references to custom information about this ink 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 ink object properties.

Ink Objects

Color

One main purpose of an ink object's existence is to specify the color of a shape. Because there is only one ink object per shape, it follows that each QuickDraw GX shape can have only one color. The only exception to this is for bitmap shapes, which use pixel values rather than an ink object to specify colors. (Picture shapes have no color at all apart from the colors of their component shapes, and thus do not use their ink object.)

The color in an ink object is defined with a `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;
};
```

The color structure specifies three characteristics of a color:

- the color's color space, which tells what kind of format the color has—such as red-green-blue (RGB), hue-saturation-value (HSV), or luminance (grayscale).
- a reference to a color profile object that contains information for converting the device-independent color in this ink object into color-corrected values on a particular output device. If the reference is `nil`, the QuickDraw GX default color profile is used.
- the numeric color values that (for the given color space) specify the color of this ink object. An individual color has one number for each dimension, or color component, in the color's color space; for example, an RGB color value consists of three color component values. A color may consist of a maximum of four components.

Ink Objects

To set and manipulate the color of an ink object requires an understanding of how color works in QuickDraw GX. The color structure, color spaces, and color profiles are all described in detail in the chapter “Colors and Color-Related Objects” in this book.

Transfer Mode

The transfer mode in an ink object is contained in a `gxTransferMode` structure:

```
struct gxTransferMode{
    gxColorSpace          space;
    gxColorSet            set;
    gxColorProfile        profile;
    Fixed                 sourceMatrix[5][4];
    Fixed                 deviceMatrix[5][4];
    Fixed                 resultMatrix[5][4];
    gxTransferFlag        flags;
    struct gxTransferComponent component[4];
};
```

Like the color structure just described, the transfer mode structure specifies a color space, and may contain a reference to a color profile object or a *color set* object, which contains an array of available colors. A transfer mode specifies its own color space because it can perform its operations according to its own definitions of color, independent of the color specifications in the rest of the ink object.

The transfer mode structure contains three 5×4 matrices (5 rows, 4 columns), the *source matrix*, *device matrix*, and *result matrix*, which it can use to transform colors for special effects, by blending proportions of the colors’ components. In addition, it contains a set of *transfer mode flags* that control several aspects of the transfer mode operation.

The structure also contains up to four *transfer components*, used along with the matrices in the transfer mode operation. Transfer components contain the actual specification of the mode of transfer to use when drawing. Transfer components are defined by the `gxTransferComponent` structure:

```
struct gxTransferComponent{
    gxComponentMode    mode;
    gxComponentFlag    flags;
    gxColorValue        sourceMinimum;
    gxColorValue        sourceMaximum;
    gxColorValue        deviceMinimum;
    gxColorValue        deviceMaximum;
    gxColorValue        clampMinimum;
    gxColorValue        clampMaximum;
    gxColorValue        operand;
};
```

Ink Objects

A transfer component contains a *component mode* specifying the type of transfer mode (like “copy” or “XOR”) to use, an operand to apply (if the type calls for an operand), a set of maximum and minimum color values, and a set of flags. There is one transfer component for each color component (dimension) in the transfer mode’s color space. Each of the transfer components in the transfer mode structure may specify a different component mode, which means that each dimension of a color space can be drawn with a different transfer mode when a shape is drawn.

How these parts of the transfer mode structure and transfer component structure define the transfer mode for drawing, and how you can use transfer modes to obtain the proper effect when drawing, are described in the section “About Transfer Modes” beginning on page 5-11.

Ink Attributes

Each ink object has a set of ink attributes, a group of flags that affect the dithering and halftoning behavior when the shape associated with the ink is drawn. *Dithering* is the use of repeating patterns of differently colored pixels to simulate colors not available in a view device’s color space. *Halftoning* is the process of representing varying color intensity with evenly spaced dots of one color (but of different sizes) separated by a background of another color. The dither level and the halftone characteristics for all drawing to a view port are specified in the view port object, but you can use an ink object’s attributes to affect them for individual shapes that use that ink.

Ink attributes allow you to turn halftoning or dithering on or off, and to affect both the number of colors used in dithering and the alignment of the patterns of dithered pixels. Table 5-1 lists the ink attribute constants and describes what each one means. The constants are defined in the `GxInkAttributes` enumeration.

Table 5-1 Ink attributes

Constant	Value	Explanation
<code>GxPortAlignDitherInk</code>	0x0001	If set, QuickDraw GX aligns the dither pattern to the view device coordinates. If this attribute is clear (the default), QuickDraw GX aligns the dither pattern to the view port coordinates.
<code>GxForceDitherInk</code>	0x0002	If set, QuickDraw GX forces the dithering operation to use exactly the number of colors specified by the view port’s dither level. If this attribute is clear, QuickDraw GX may use fewer colors (in a simpler dither pattern) when constructing the dither pattern.

continued

Ink Objects

Table 5-1 Ink attributes (continued)

Constant	Value	Explanation
<code>gxSuppressDitherInk</code>	0x0004	If set, QuickDraw GX ignores the view port dither level, if any, and draws without dithering.
<code>gxSuppressHalftoneInk</code>	0x0008	If set, QuickDraw GX ignores the view port halftone, if any, and draws without creating a halftone.

IMPORTANT

Make sure that the `gxPortAlignDitherInk` attribute is cleared in ports associated with windows, so that if the window is dragged, updates using dithered drawing will match the existing parts of the drawing. (The attribute is clear by default.) ▲

Dithering, dither level, and halftones are described in more detail in the chapter “View-Related Devices” in this book.

The Default Ink Object

When QuickDraw GX first creates an ink object, that object has default characteristics defined by QuickDraw GX. A default ink object has the following properties:

- No attributes set.
- An empty tag list.
- An owner count of 1.
- Color space set to `gxRGBSpace` with each color component set to 0, which represents black in this color space.
- Transfer mode set to `gxCopyMode`, with identity transfer mode matrices, color limits of 0 to 0xFFFF, and all flags cleared. Copy mode is the default transfer mode assigned to all color components of an ink object, because it is most common and fastest.

Transfer modes, matrices, color limits, and flags are described in subsequent sections of this chapter. Color spaces and color components are described in the chapter “Colors and Color-Related Objects” in this book.

To reset an ink object to its default properties, use the `GXResetInk` function, described on page 5-60.

About Transfer Modes

Basically, ink objects exist to specify two important characteristics of a shape: its color and the transfer mode to draw it with. Colors are described in the chapter “Colors and Color-Related Objects” in this book. Transfer modes are described here.

Transfer modes specify how a shape’s color is transferred onto a device. The color of a shape to be drawn (the *source color*) interacts with the existing color (the *destination color*) on the device it is drawn to. The color that results from that interaction is called the *result color*. The result color is the color of the destination after the drawing occurs.

Note that colors from different color spaces can be used. The source and destination colors are converted to the transfer mode’s color space, and the resulting color is then reconverted to the destination color space.

Bitmaps and the ink object

A bitmap shape does not use the color in its ink object, but it does use the ink’s transfer mode. Transfer modes work the same for the pixels of bitmaps as they do for colors in ink objects. For more information, see the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. ♦

QuickDraw GX allows you to influence the transfer mode operation in very flexible and powerful ways. By manipulating different parts of the transfer mode structure, you can specify

- the type of transfer mode to apply to each color component
- the color space in which to perform the transfer-mode calculations
- limits on the values of color components that can be permitted in the source, destination, or result colors
- values for the source, destination, or result matrices that can allow you to perform sophisticated transformations within and across color components
- values for flags that affect several aspects of the transfer mode operation

The rest of this section discusses these five aspects of transfer modes. The section concludes with a summary diagram (Figure 5-18 on page 5-37) of the transfer mode process.

Transfer Mode Types

Transfer modes can be specified by type, also called *component mode*. Transfer mode types in QuickDraw GX are called component modes because QuickDraw GX allows each color component to have its own transfer mode type. In RGB color space, for example, the red component of the color may be drawn with a different transfer mode type than the blue component.

Ink Objects

QuickDraw GX supports several conceptual categories of component modes:

- arithmetic
- Boolean
- pseudo-Boolean
- highlight
- alpha-channel

The characteristics of and most typical uses for the component modes within each category are summarized in the following subsections.

Copy mode is the default

Even though QuickDraw GX supports 18 different component modes, most applications in most situations need only one, an arithmetic mode called *copy mode*. In copy mode, the source color completely replaces the destination color. Copy mode is the default transfer mode in QuickDraw GX; therefore, you need information about other transfer modes only if you want them for special effects. ♦

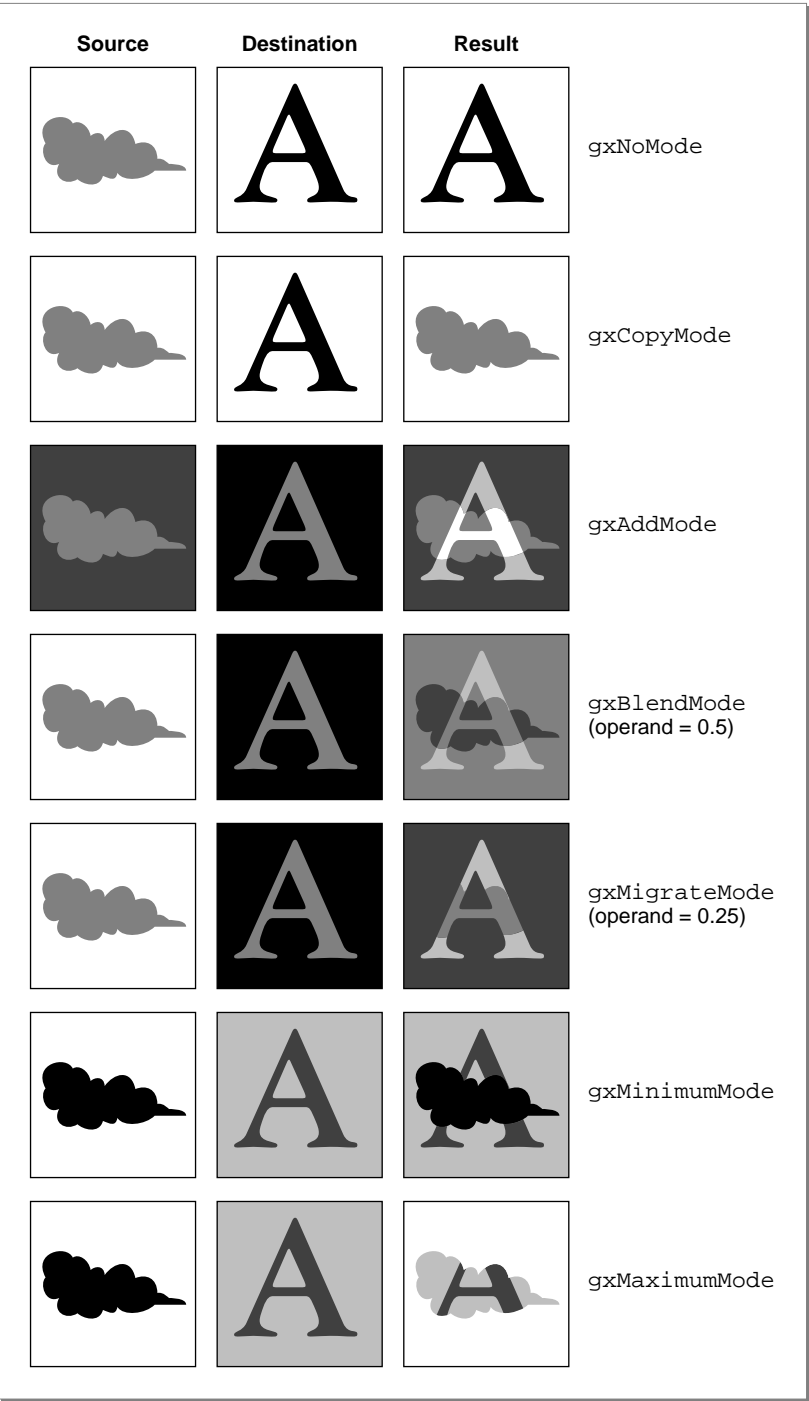
Arithmetic Transfer Modes

In arithmetic transfer modes, the numerical values of source and destination for a color component are combined arithmetically to determine the result value for that color component. In most color spaces, a color component value can vary from 0 (no intensity) to 0xFFFF (maximum intensity). You can also use the constant `gxColorValue1` to represent maximum intensity (0xFFFF).

Figure 5-2 shows examples of drawing with the arithmetic transfer modes. In each case, the source image (left) combines with the destination image (center) to produce the result image (right). You can think of the images either as two bitmaps, or as two source shapes (cloud and background) that are drawn over two destination shapes (letter and background).

Each example shows how transfer mode affects drawing within a single color component (reflected as shades of gray in the figure, where black equals 0 and white equals 0xFFFF). The constant that specifies the transfer mode type is shown to the right of each example. Note also that two of the arithmetic transfer modes use an *operand*, a numerical value that affects the outcome of the transfer-mode operation.

Figure 5-2 Arithmetic transfer modes



Ink Objects

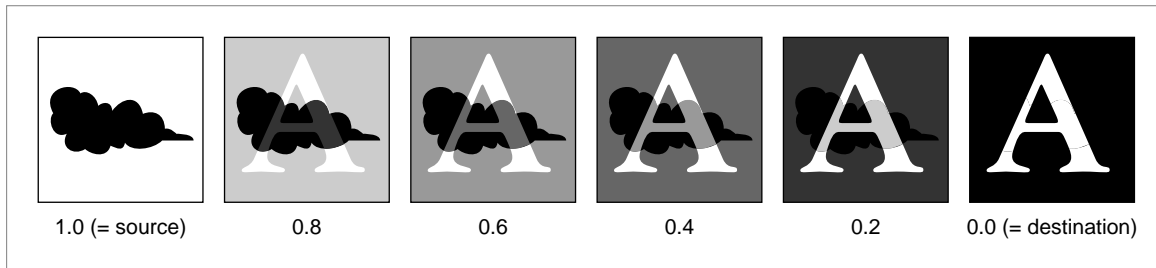
The constants that define transfer mode type are defined in the `gxComponentModes` enumeration. The arithmetic modes have the following values and meanings:

Constant	Value	Explanation
<code>gxNoMode</code>	0	No mode. No transfer occurs. For this component of the color, the destination is left as it was. This mode is useful for suppressing drawing when certain logical conditions are met, or for not drawing one color component while allowing other components to be drawn.
<code>gxCopyMode</code>	1	Copy mode. The source color component is copied to the destination. The destination component is ignored. This is the most common transfer mode, and is the default for QuickDraw GX.
<code>gxAddMode</code>	2	Add mode. The source color component is added to the destination component, but the result is not allowed to exceed the maximum value (0xFFFF or <code>gxColorValue1</code> ; white in Figure 5-2).
<code>gxBlendMode</code>	3	Blend mode. The result is the average of the source and destination color components, weighted by a ratio specified by the operand component (0.5 in Figure 5-2). The operand varies from 0 (all destination) to 0xFFFF or <code>gxColorValue1</code> (all source), although it is customary to interpret it as varying between 0 and 1.
<code>gxMigrateMode</code>	4	Migrate mode. The destination color component is moved toward the source component by the value of the step specified in the operand component (0.25, or 0x4000 in Figure 5-2). Migrate mode is similar to blend mode, except that the change in destination component is an absolute amount, rather than a proportion of the difference between it and the source component. If the source has a greater color component value than the destination, the migration is positive; if the destination has a greater value than the source, the migration is negative. In either case, the amount of migration cannot be greater than the difference between the destination and the source values.
<code>gxMinimumMode</code>	5	Minimum mode. The source component replaces the destination component only if the source component has a smaller value. (In Figure 5-2, drawing occurs only within the area occupied by the cloud.)
<code>gxMaximumMode</code>	6	Maximum mode. The source component replaces the destination component only if the source component has a larger value. (In Figure 5-2, drawing occurs only outside of the area occupied by the cloud.)

Ink Objects

The operand parameter is used by blend mode to specify the ratio of source and destination component. It is used by migrate mode to specify the step size by which the destination component moves toward the source component. Figure 5-3 shows examples of the result of drawing with blend mode, using several different values for the operand. (Color Plate 1 at the front of this book shows the same example in color.)

Figure 5-3 Blend example with different operand values

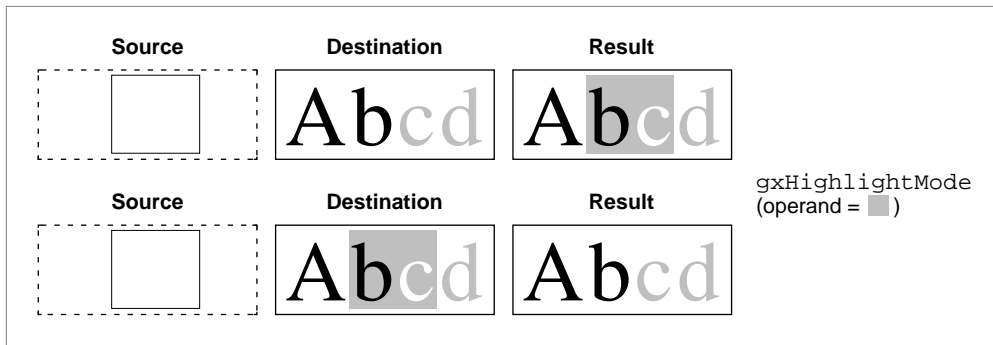


Highlight Transfer Mode

The highlight transfer mode is used for highlighting in color applications. It is most commonly used to draw (and clear) a colored rectangle around a selection, without altering the color of the item or items selected. In text, it gives the effect of drawing over the letters with a highlighting pen.

Like some of the arithmetic transfer modes, highlight mode uses an operand to control the outcome of the highlighting operation. Highlight mode operates by replacing the source color with the operand color, and the operand color with the source color, in the destination.

The upper row of images in Figure 5-4 shows a simple example of the application of highlight mode. The operand value is represented with shading rather than as a number, to illustrate how its color affects colors in the image. The source shape is a white rectangle that is drawn over the two middle letters in the destination image. (The gray letters in the line of text in the destination image represent the same color-component value as the operand, and the white area around the letters in the destination represents the same color-component value as the source.)

Figure 5-4 Highlight transfer mode

Note that black in the destination is unaffected, whereas white becomes gray and gray becomes white. A single constant specifies highlight mode, with the following value and meaning:

Constant	Value	Explanation
<code>gxHighlightMode</code>	7	Highlight mode. The source component and operand component are swapped in the destination. Other components in the destination are ignored.

In highlight mode, the source color can be thought of as the “background” color that is to be highlighted, and the operand color is the color of the highlighting pen. As the lower set of images in Figure 5-4 shows, redrawing a highlighted selection causes the source and operand colors to swap once more, effectively removing the highlighting.

The operand for highlight mode is a normal color component value that varies from 0 (no intensity) to the maximum intensity permitted for that component (normally `0xFFFF`, or `gxColorValue1`).

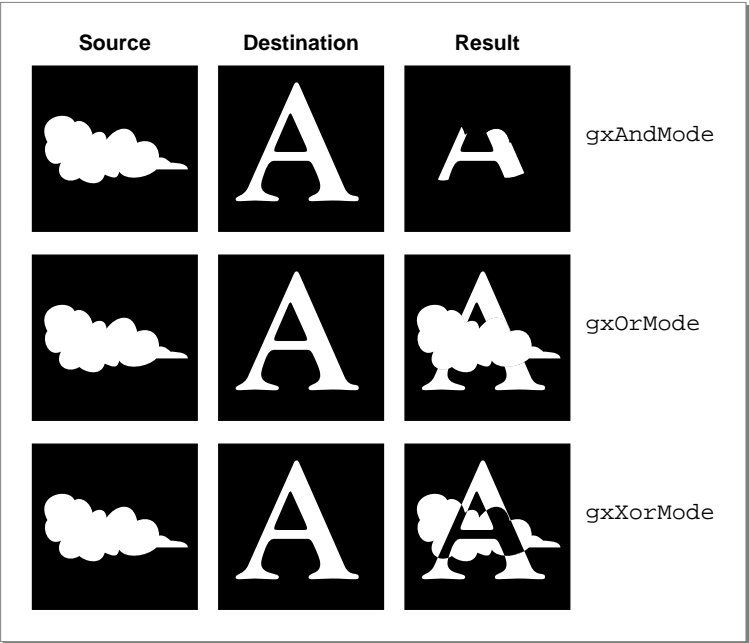
QuickDraw GX applies highlight mode only if all components in the color space specify it. An error occurs if some components specify highlight mode and others do not.

Boolean Transfer Modes

In Boolean transfer modes, the result value for a color component is determined by bit operations performed on the source and destination component values. Boolean transfer modes are most common in black-and-white drawing; in any bit depth other than 1, they yield results that can be difficult to predict because they depend on the states of the individual bits in each color-component value.

Figure 5-5 shows examples of drawing with the Boolean transfer modes at a bit depth of 1. In each case, the source image combines with the destination image to produce the result image. In these examples, black represents a bit value of 0 (clear), and white represents a bit value of 1 (set). The constant that specifies the transfer mode type is shown to the right of each example.

Figure 5-5 Boolean transfer modes (1-bit depth)



The Boolean modes have the following values and meanings:

Constant	Value	Explanation
gxAndMode	8	AND mode. The bits of the source color and destination color are combined using an AND operation. Only bits that are set in both source and destination remain set in the result.
gxOrMode	9	OR mode. The bits of the source color and destination color are combined using an OR operation. Bits that are set in either the source or the destination or in both are set in the result.
gxXorMode	10	XOR mode. The bits of the source color and destination color are combined using an exclusive-OR (XOR) operation. Bits that are set in the source but not the destination, and bits that are set in the destination but not the source, are set in the result. All other bits are cleared in the result.

Ink Objects

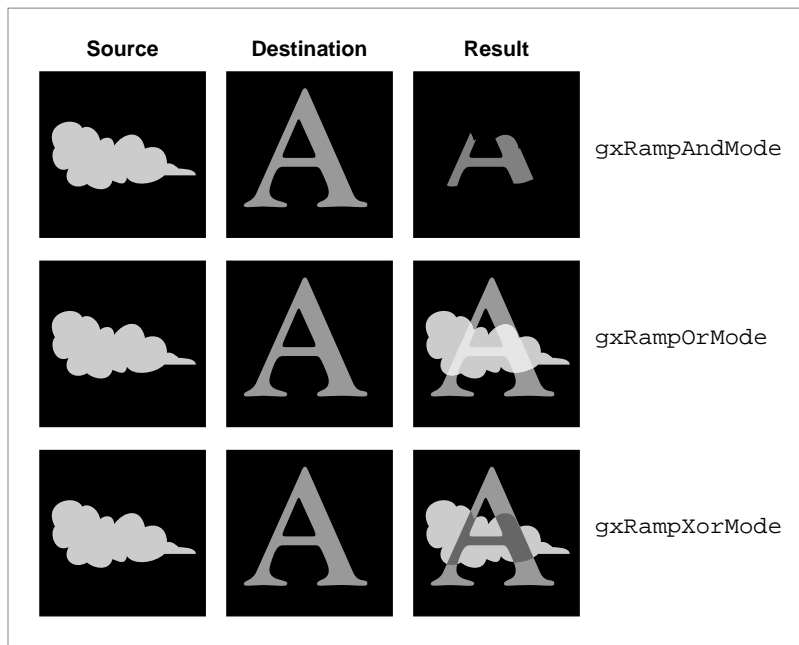
Even though they are most easily explained in terms of single-bit depths, Boolean modes are not restricted to 1-bit drawing. They can be used with any kind of color values, and are useful for manipulating colors in an indexed color space.

Pseudo-Boolean Transfer Modes

In pseudo-Boolean transfer modes, the result value for a color component is determined by normalizing the source and destination values and performing a simple arithmetic operation, to achieve consistent and predictable results analogous to 1-bit Boolean operations.

Figure 5-6 shows examples of drawing with the pseudo-Boolean transfer modes. In each case, the source image combines with the destination image to produce the result image. The constant that specifies the transfer mode type is shown to the right of each example.

Figure 5-6 Pseudo-Boolean transfer modes



Ink Objects

The constants for the pseudo-Boolean component modes have the following values and meanings:

Constant	Value	Explanation
<code>gxRampAndMode</code>	11	Ramp-AND mode. The source and destination color components are treated as ranging from 0 to 1; their product ($\text{source} \times \text{destination}$) is returned.
<code>gxRampOrMode</code>	12	Ramp-OR mode. The source and destination color components are treated as ranging from 0 to 1; the result of $(\text{source} + \text{destination} - \text{source} \times \text{destination})$ is returned.
<code>gxRampXorMode</code>	13	Ramp-XOR mode. The source and destination color components are treated as ranging from 0 to 1; the result of $(\text{source} + \text{destination} - 2 \times \text{source} \times \text{destination})$ is returned.

Note that the pseudo-Boolean and Boolean modes are similar in several ways:

- The mode `gxRampAndMode` is similar to `gxAndMode` in that nonzero values occur in the result only where both source and destination are nonzero.
- The mode `gxRampOrMode` is similar to `gxOrMode` in that nonzero values occur in the result wherever either the source or the destination is nonzero.
- The mode `gxRampXorMode` is similar to `gxXorMode` in that the result is close to zero wherever the source and destination are close to each other in value.

The difference between the pseudo-Boolean and Boolean modes is that, for multi-bit pixel depths, the results for `gxRampAndMode`, `gxRampOrMode`, and `gxRampXorMode` are predictable and vary smoothly and continuously with component intensity. For 1-bit depths, these modes are identical to their Boolean equivalents.

The pseudo-Boolean modes are commonly used as component modes for alpha channels in color spaces that have an alpha channel. See “Alpha-Channel Transfer Modes” (next).

Alpha-Channel Transfer Modes

Several QuickDraw GX color spaces (`gxRGBASpace`, `gxARGB32Space` and `gxGrayASpace`) have an *alpha channel*. This is an additional color component that controls the opacity or transparency of a color. For example, a red pixel in a source image can be completely opaque, in which case it typically retains its red color when drawn over a blue pixel in the destination image. Or, the pixel can be completely transparent, in which case it typically loses all its color and turns totally blue when drawn over a blue pixel. Or, it can have an opacity of, say, 0x7FFF (50%), in which case it typically turns magenta when drawn over a blue pixel.

Alpha channel values can be used to allow parts of one image to show through “holes” in another, to show translucency in objects that are drawn over other objects, and to perform anti-aliasing (smoothing of jagged edges) by giving feathered, semi-transparent borders to opaque objects.

When assigning transfer modes to colors with an alpha channel, you typically use two different kinds of modes:

- To get the proper result color for each color component, you use an alpha-channel transfer mode. These modes take alpha-channel values into account when calculating result values for the color components.
- To get the proper result opacity for the alpha channel itself, you typically use an arithmetic or pseudo-Boolean transfer mode.

This section describes how the different modes within each category work to give you the results you want.

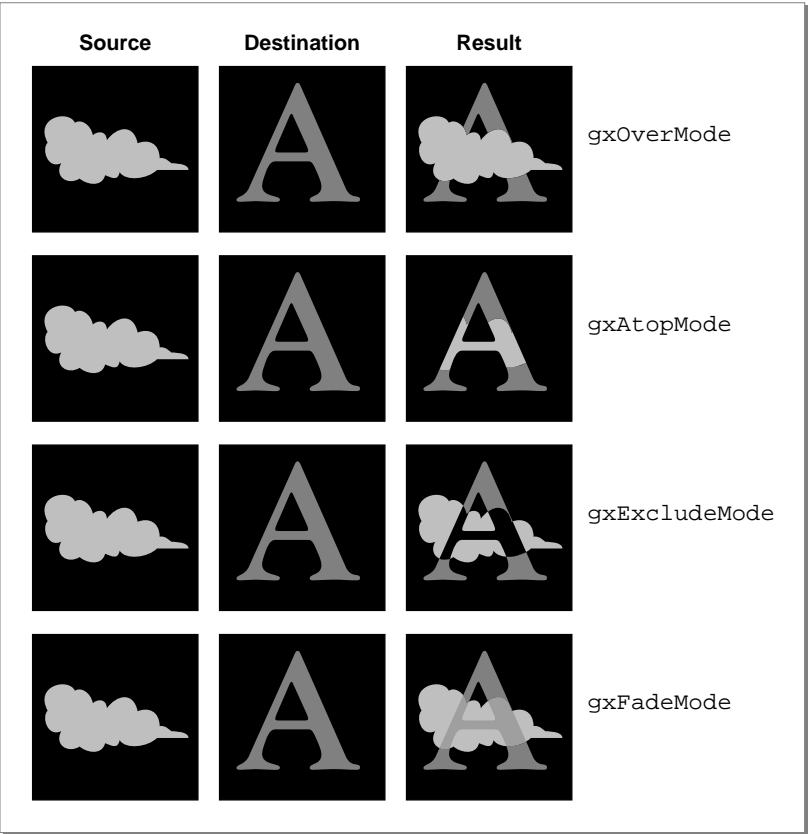
Modes for the Color Components

Figure 5-7 shows examples of how values for a color component might be calculated, given a source image and a destination image consisting of objects (or pixels) that differ in opacity. In each example, the source image (an opaque, light gray cloud against a transparent black background) combines with the destination image (an opaque, dark gray “A” on a transparent black background), to form the result image. The constant that specifies the transfer mode type is shown to the right of each example.

Ink Objects

Each example shows how the alpha-channel transfer mode affects drawing within a single color component. The mode takes into account not only the source and destination color components, but the source and destination opacities as well.

Figure 5-7 Alpha-channel transfer modes



Ink Objects

The constants for the alpha-channel component modes have the following values and meanings:

Constant	Value	Explanation
<code>gxOverMode</code>	14	Over mode. The source color is copied to the destination, and the source transparency controls where the destination color shows through. Where both are transparent, no drawing occurs (result equals destination).
<code>gxAtopMode</code>	15	Atop mode. The source color is placed over the destination, but the resulting destination retains the original destination's transparency. The effect is that opaque parts of the source are clipped to cover only opaque parts of the destination.
<code>gxExcludeMode</code>	16	Exclude mode. The destination color remains visible only where the source is transparent, and the source color is copied anywhere the destination is transparent. Where both are transparent, no drawing occurs (result equals destination); where both are opaque, the result color is 0 (no intensity).
<code>gxFadeMode</code>	17	Fade mode. The source is blended with the destination, using the relative alpha values as the ratio for the blend. Where both are transparent, the result is the average of the source and the destination).

As Figure 5-7 shows, the `gxOverMode` mode is similar to the arithmetic transfer mode `gxCopyMode`, except that it allows for transparency in the source image. Likewise, the `gxAtopMode` mode is similar to `gxCopyMode`, but it preserves the transparency of the destination image by clipping the opaque source to the destination image. The `gxExcludeMode` mode is somewhat like the Boolean transfer mode `gxXorMode`, in that opaque parts of each image appear only where the other is not opaque. The `gxFadeMode` mode is like the arithmetic `gxBlendMode`, except that the operand that controls the blend ratio is determined by the relative opacities.

Note that the images shown in Figure 5-7 are very simple and their opacities are either 0 (completely transparent) or `gxColorValue1` (completely opaque). Because an alpha component can have a wide range of partial opacities, very sophisticated translucency and color-blending effects are possible, as well as the simple masking effects shown here.

The exact formulas for determining result color are the following. In these formulas, *sA* = source alpha-channel value; *dA* = destination alpha-channel value; *sC* = source color-component value; *dC* = destination color-component value; and *rC* = result color-component value.

- For `gxOverMode`:

$$rC = (sA \times (sC - dA \times dC) + dA \times dC) / (sA + dA - sA \times dA)$$

- For `gxAtopMode`:

$$rC = dC + sA \times (sC - dC)$$

Ink Objects

■ For `gxExcludeMode`:

$$rC = (sA \times sC + dA \times dC - sA \times dA \times (sC + dC)) / (sA + dA - sA \times dA)$$

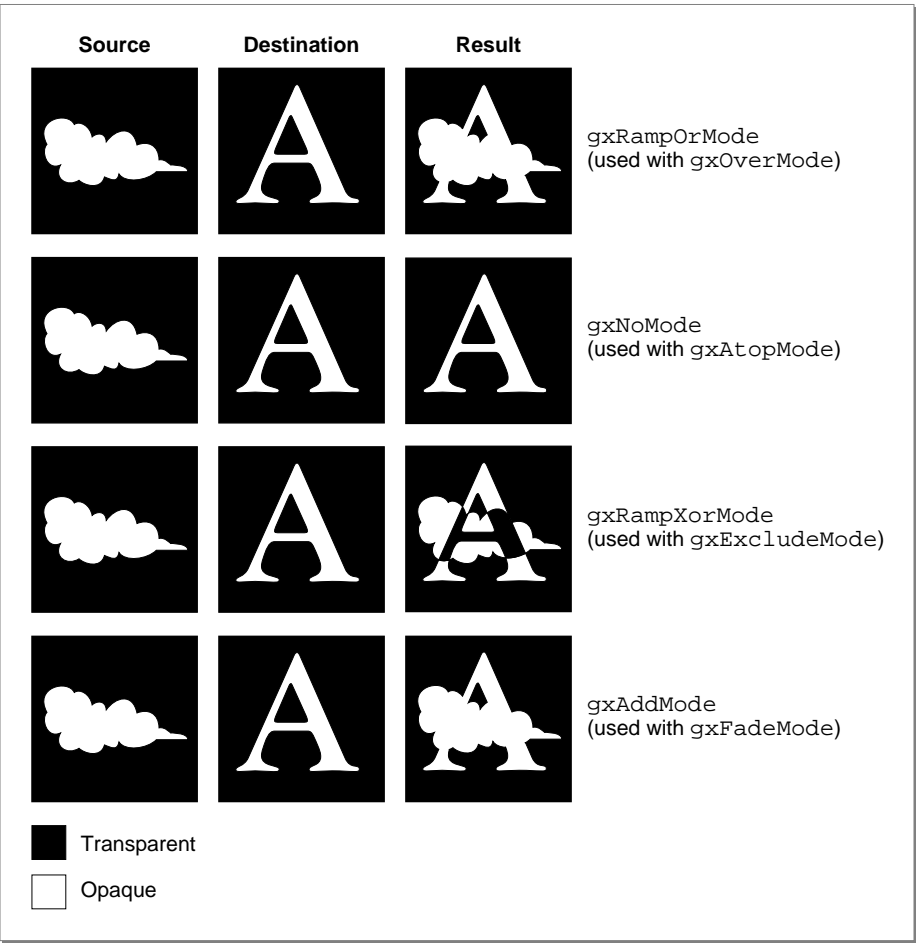
■ For `gxFadeMode`:

$$rC = sA \times sC + dA \times dC / (sA + dA)$$

Modes for the Alpha Channel

For calculating the result value for the alpha channel itself, you typically use one of the arithmetic or pseudo-Boolean transfer modes presented in the previous sections—one that takes into account only the source and destination opacities. Figure 5-8 shows typical modes used and their effects on opacity, using the same images as those presented in Figure 5-7. In this figure, black represents complete transparency and white represents complete opacity. (If alpha-channel values between the two extremes existed in these examples, they would be shown in shades of gray.)

Figure 5-8 Typical modes used to determine result opacity for the alpha channel



Ink Objects

Note from Figure 5-8 that the mode you use to determine the result opacity of the alpha channel usually depends on what alpha-channel mode you use to get color-component values:

- Use `gxRampOrMode` to calculate result alpha-channel values if you want the opacities of both source and destination summed proportionally (in a pseudo-Boolean manner; see the description of `gxRampOrMode` on page 5-19) to achieve a result opacity. Thus, if you use `gxOverMode` for the color-components, you would typically use `gxRampOrMode` for the alpha channel.
- Use `gxNoMode` to calculate result alpha-channel values if you want the opacity of the destination to remain unchanged. Thus, if you use `gxAtOpMode` for the color-components, you would typically use `gxNoMode` for the alpha channel.
- Use `gxRampXorMode` to calculate result alpha-channel values if you want a maximum result opacity where there is a maximum difference in opacities between source and destination. Thus, if you use `gxExcludeMode` for the color-components, you would typically use `gxRampXorMode` for the alpha channel.
- Use `gxAddMode` to calculate result alpha-channel values if you want the result opacity to reflect the sum of the opacities of the source and destination (pinned to the maximum permitted value). Thus, if you use `gxFadeMode` for the color-components, you would typically use `gxAddMode` for the alpha channel.

Note

When converting a color from a color space that does not have an alpha channel to one that does, QuickDraw GX sets the alpha channel intensity to maximum (opaque). When a color is converted from a color space that does have an alpha channel to one that does not, the alpha channel is lost. ♦

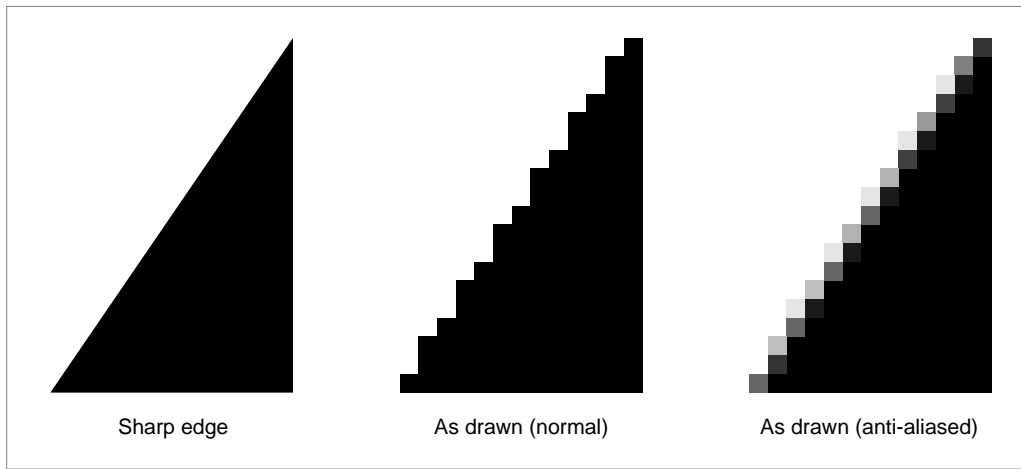
Transparency Ramps and Anti-Aliasing

Two common applications for alpha-channel colors involve making objects or images partially opaque to give a translucent effect, and smoothing jagged edges on objects drawn at low resolution.

You can create a bitmap in which the alpha-channel values of the pixels vary smoothly in one or more directions, thus creating a transparency ramp that allows the destination image to show through the source image to varying degrees across the bitmap. Color Plate 2 at the front of this book, for example, shows the kind of effect that can be achieved with a simple alpha-channel ramp.

The smoothing of jagged edges on displayed objects is called *anti-aliasing*. You can perform anti-aliasing by modifying the alpha-channel values of the pixels surrounding the edges of an opaque object. You make an individual pixel more or less opaque, based on the proportion of that pixel that the object is computed to cover.

In Figure 5-9, for example, the left image shows the computed position of the edge of a shape in a bitmap. The center image shows how that edge is displayed normally, given the resolution of the bitmap. The right image shows that edge as it might be displayed with anti-aliasing applied. The apparent jaggedness is decreased because pixels near the edge allow the background to show through to varying degrees.

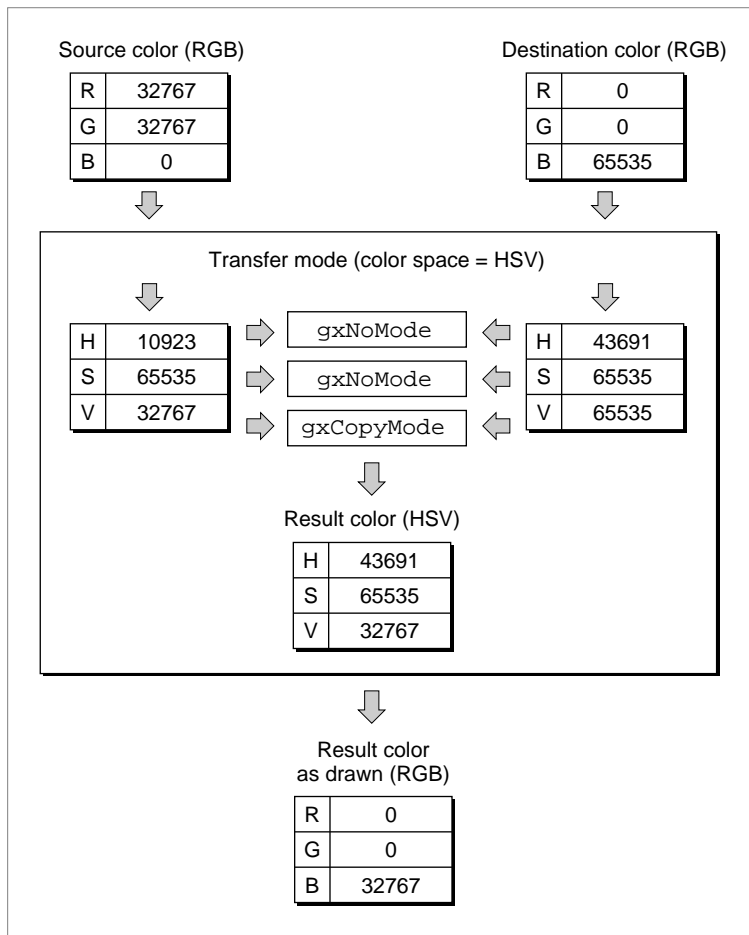
Figure 5-9 Anti-aliasing

Transfer Mode Color Space

The `space` field in the transfer mode structure specifies the color space the transfer mode calculations take place in. This space does not need to be the same as the color space specified by the ink's color, or the destination color space as specified by the view port or view device with which the ink is associated. The source and destination colors are converted into the color space that provides the context for the transfer, and the resulting color is then reconverted to the destination color space. Keep in mind that all transfer mode computations take place in the transfer mode's color space.

You needn't convert color values among different spaces yourself in order to use a different transfer mode. The transfer mode operation automatically converts colors from the ink's color space and the view device's color space, manipulates them, and then converts the result color back to the view device's color space for drawing. In creating shapes, you can work in whatever color space is convenient for you; when drawing, you can use any transfer mode color space you want; and neither color space need be the same as the color space used by the view device to which you are drawing.

Figure 5-10, for example, shows a source color in RGB space as specified in an ink object, a destination color in RGB space as specified by a monitor, and a transfer mode color space of HSV, as specified by the application. The component modes selected mean that the hue and saturation of the destination are preserved, but the value (lightness) of the source is maintained. QuickDraw GX automatically performs all necessary conversions.

Figure 5-10 Automatic conversion of color values during a transfer mode operation

The transfer mode color space defines how many components are required to perform the transfer mode operation. Monochromatic (grayscale) color spaces and indexed color space require only one component to be filled out. Alpha-channel spaces and CMYK space require four components to be filled out. All other spaces require three components to be filled out. (Color spaces are described in the chapter “Colors and Color-Related Objects” in this book.)

Ink Objects

Remember that if the transfer mode's color space is `gxIndexedSpace`, the transfer mode structure must contain a reference to a color set object.

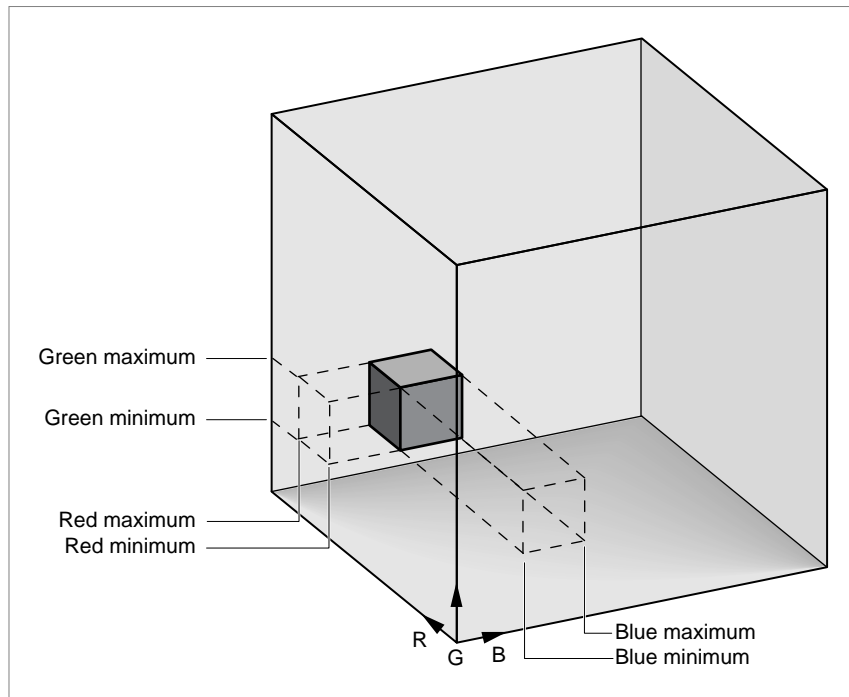
Note

Choosing a different color space can radically affect the behavior of a transfer mode. For example, if your transfer mode uses RGB space, and you have specified `gxCopyMode` for the component mode of `component[0]` and `gxNoMode` for the other components in the transfer mode structure, drawing will transfer only the red component of your source image to the destination, and leave the blue and green components of the destination as they are. If you then change the transfer mode color space to HSV and redraw, all hues in your source image will be transferred to the destination, but with the brightnesses and saturations of the original destination image. ♦

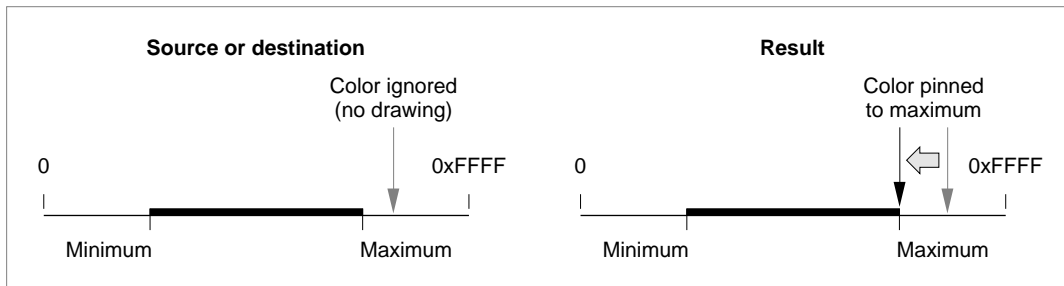
Color Limits

Transfer mode operations allow you to specify limits on the acceptable input values for the source or destination color, and on the acceptable output values for the result color. For example, in converting CMYK color to RGB, you may wish to limit the intensities to values that can be displayed without oversaturating the phosphors on a monitor's screen. Or, to create a special effect, you may want to draw only the extreme light and dark portions of an image, leaving out its midrange entirely.

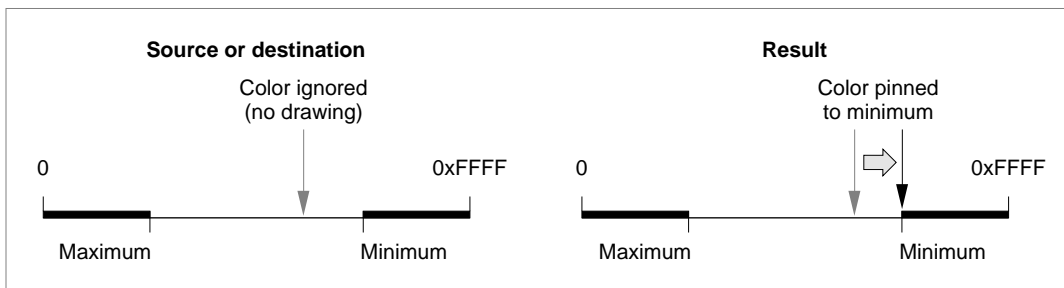
Each color component in the `component` field of the transfer mode structure can have a maximum and a minimum permitted value. The permissible ranges can be interpreted as shown in Figure 5-11. In the figure, the large cube represents all of RGB space; the small cube represents one possible example of the limits that could be imposed on allowable values for all three components.

Figure 5-11 Maximum and minimum color-component values in RGB space

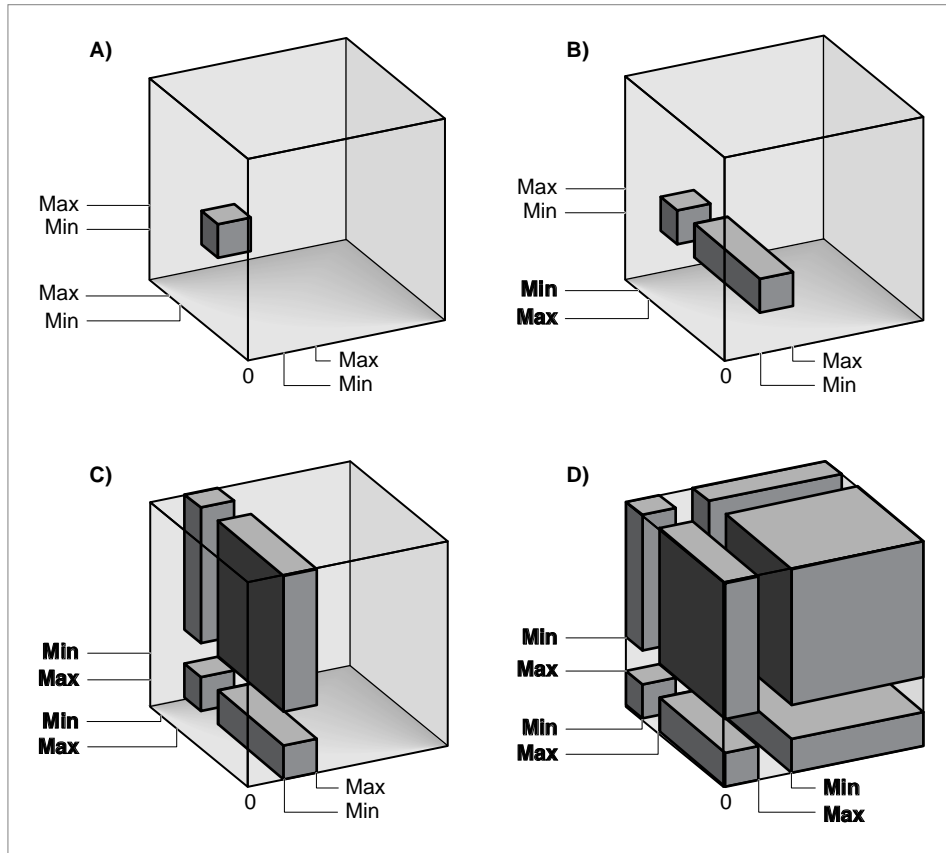
In the case of source and destination colors, color values outside the range of acceptable values (that is, outside the small cube in Figure 5-11) are ignored; if any single component value is outside of its acceptable range, no drawing occurs at all for that color. In the case of the calculated colors that result from a given transfer mode operation, color values outside of the acceptable range are *pinned* to, or moved so that they don't exceed, the nearest acceptable value (the closest edge of the small cube). See Figure 5-12.

Figure 5-12 How minimum and maximum color limits affect drawing

For a given component, the maximum value for a color limit can be either greater or smaller than the minimum. If the maximum is less than the minimum, only the extreme color values (that is, values *outside* of the small cube area in Figure 5-11) are allowed. See Figure 5-13.

Figure 5-13 How reversed minimum and maximum color limits affect drawing

Each of the components in a color space can have its limits set entirely independently of the others. Figure 5-14 shows the effects of reversing, in turn, the maximum and minimum values for each of the three axes in RGB space.

Figure 5-14 The effects of reversing maximum and minimum in a color space

Where the words *Min* and *Max* are bold in Figure 5-14, the minimum is greater than the maximum. Refer to Figure 5-11 on page 5-28 for the positions of the color axes on the RGB cube in this figure:

- In drawing (A), all minimum limits are less than their respective maximums; the allowable color ranges form a small cube, just as in Figure 5-11.
- In drawing (B), the maximum on the red axis is less than the minimum; only red color values outside of the range of the small cube are permitted, whereas blue and green must still be within the limits of the small cube. The acceptable color values form two rectangular solids within RGB space.

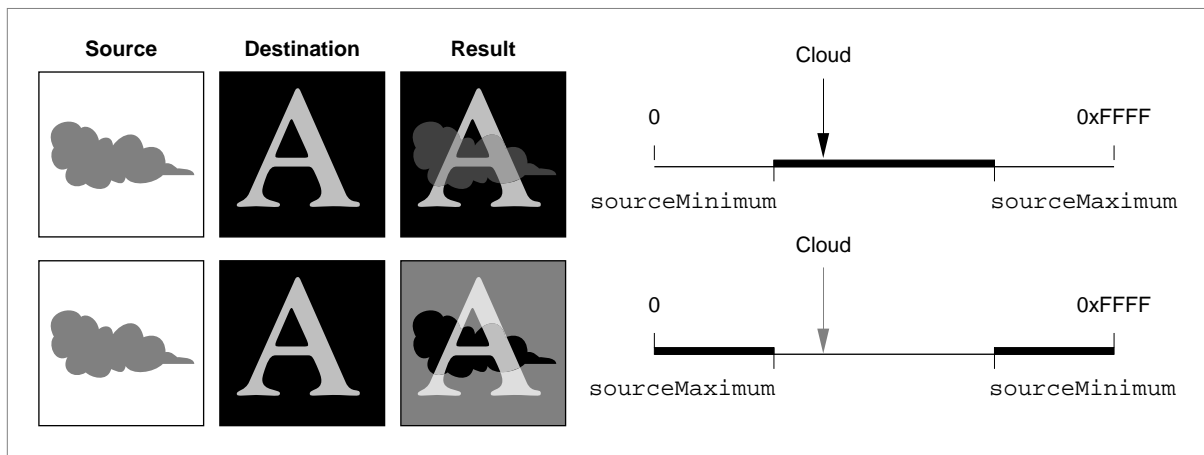
Ink Objects

- In drawing (C), the maximum and minimum on the green axis are also reversed; the acceptable color values form a more complicated set of solids.
- In drawing (D), all maximum and minimum color limits are reversed. In that case, only color values at the outer corners of the color space (all components outside of the range of the small cube) are acceptable.

Source Color Limits

The `sourceMinimum` and `sourceMaximum` fields in a color component's `gxTransferComponent` structure define the allowable range of values for source color in that component. Color values outside of the range cause no drawing to occur. If `sourceMaximum` is less than `sourceMinimum`, the range allowed consists of values less than `sourceMaximum` or greater than `sourceMinimum`. Figure 5-15 shows the effect of `sourceMinimum` and `sourceMaximum` on drawing using blend mode.

Figure 5-15 The effect of source color limits on drawing

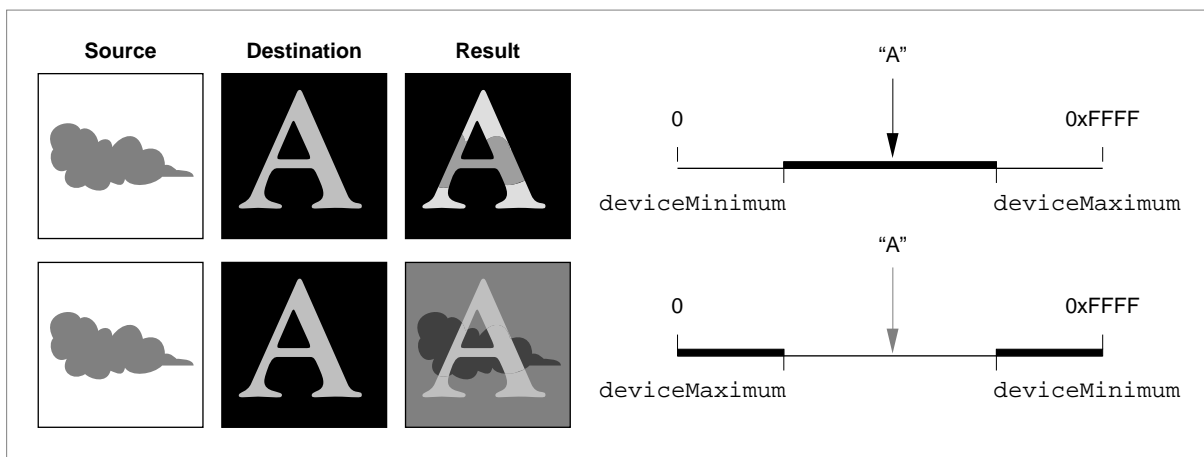


Note in Figure 5-15 that, when `sourceMinimum` is less than `sourceMaximum`, only the cloud in the source image is within the source limits, so only the cloud is blended with the destination image to create the result. Conversely, when `sourceMaximum` is less than `sourceMinimum`, the cloud in the source image is outside the source limits, so it is the only part of the source that is *not* blended with the destination image when creating the result.

Destination Color Limits

The `deviceMinimum` and `deviceMaximum` fields in a color component's `gxTransferComponent` structure define the allowable range of values for destination color in that component. Destination color values outside of the range cause no drawing to occur for that color. If `deviceMaximum` is less than `deviceMinimum`, the range allowed consists of values less than `deviceMaximum` or greater than `deviceMinimum`. Figure 5-16 shows the effect of `deviceMinimum` and `deviceMaximum` on drawing using blend mode.

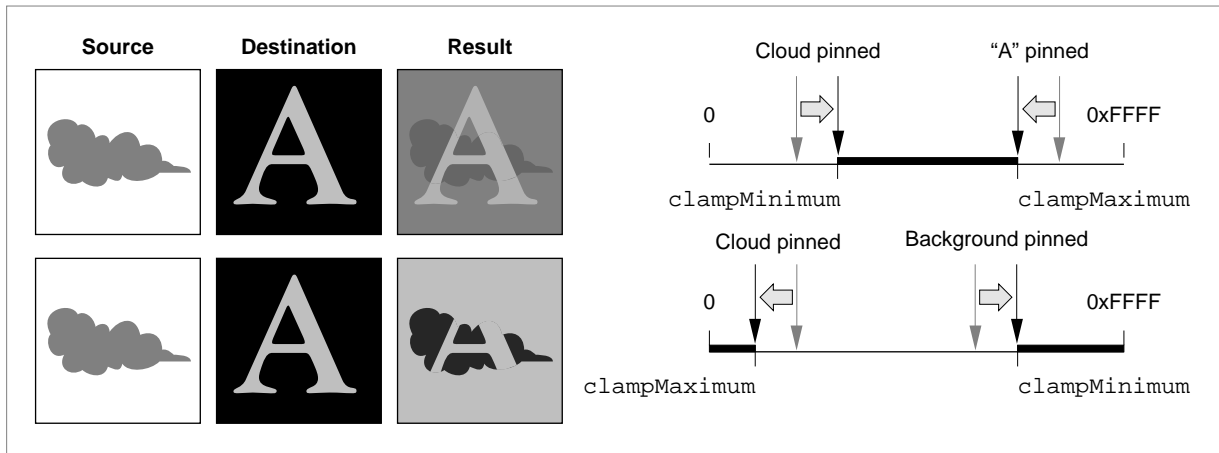
Figure 5-16 The effect of destination color limits on drawing



Note in Figure 5-16 that, when `deviceMinimum` is less than `deviceMaximum`, only the letter "A" in the destination image is within the destination limits, so the source is blended with the destination image only within the limits of the "A" to create the result. Conversely, when `deviceMaximum` is less than `deviceMinimum`, the "A" is outside the destination limits, so it is the only part of the destination *not* blended with the source to create the result.

Result Color Limits

The `clampMinimum` and `clampMaximum` fields in a color component's `gxTransferComponent` structure define the allowable range of values for the result color in that component. Color values outside of the range are pinned to the nearest clamp limit. If `clampMaximum` is less than `clampMinimum`, the range allowed consists of values less than `clampMaximum` or greater than `clampMinimum`. Figure 5-17 shows the effect of `clampMinimum` and `clampMaximum` on drawing using blend mode.

Figure 5-17 The effect of result color limits on drawing

Note in Figure 5-17 that, when `clampMinimum` is less than `clampMaximum`, extreme color values cannot occur in the result. The portions of the “A” outside of the cloud are darker than they would normally be with blend mode, and the portions of the cloud outside of the letter are lighter than they would normally be. Conversely, when `clampMaximum` is less than `clampMinimum`, midrange values are not possible in the result. The background in the result is lighter than it would normally be with blend mode, and the portions of the cloud outside of the “A” are darker than they would normally be.

Note

Pinning restricts the value of the computation, not necessarily the value allowed for the actual pixel. The pixel value is the closest found to the computation, which may be outside of the range of `clampMinimum` and `clampMaximum`. ♦

Transfer Mode Matrices

QuickDraw GX provides three matrices in the transfer mode structure to give you great freedom in controlling, modifying, and combining source, destination, and result color components when performing a transfer mode operation.

The *source matrix*, *device matrix*, and *result matrix* provide a way of scaling, weighting, swapping, and averaging the components of a color space before or after the transfer mode operation. Each matrix is a 5×4 array that specifies the mixture of each of the (up to 4) components, plus an offset.

Ink Objects

An *identity matrix*, one that has values of 1.0 along the diagonal and zero values elsewhere, has no effect. Here it is applied to a color in CMYK space:

$$\begin{bmatrix} c & m & y & k & 1 \end{bmatrix} \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} = \begin{bmatrix} c & y & m & k \end{bmatrix}$$

The values for all color components after the matrix multiplication are the same as before. All transfer mode matrices in the default ink object are identity matrices.

The bottom row of the matrix specifies an offset value. The following matrix replaces c with $1/2 c + 1/2 m$; it also scales k by 0.8 and adds 0.2 to it:

$$\begin{bmatrix} c & m & y & k & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0.0 & 0.0 & 0.0 \\ 0.5 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.0 & 0.2 \end{bmatrix} = \begin{bmatrix} (0.5c + 0.5m) & m & y & (0.8k + 0.2) \end{bmatrix}$$

The source and device matrix are applied before the transfer mode calculation and after applying source minimum and source maximum. The result of the transfer mode calculation is run through the result matrix. The use of matrices allows you to apply sophisticated mapping operations—analogueous to the scaling, rotation, translation, and distortion of shapes discussed in the chapter “Transform Objects” in this book—to the colors involved in a transfer mode operation. Matrices are also used to create color separations, and to map source color ranges to spot colors.

Note

Although color components are described by unsigned shorts (16-bit positive numbers), the math internal to transfer modes is performed with longs (32-bit signed numbers) to minimize overflow or roundoff error. As an example, elements in the source matrix could multiply by a large number, and elements in the result matrix could divide by a large number, without creating an overflow condition. ♦

Flags

QuickDraw GX provides two sets of flags in the transfer mode structure that control certain aspects of the transfer mode operation. One set operates on individual color components; the other set operates on the source or destination color as a whole, taking into account all of the components.

Transfer Component Flags

The transfer component flags are a set of flags in the `gxTransferComponent` structure (in the `component` field of the transfer mode structure) that alter the source, destination, or result value for an individual color component. There are two constants for these flags, defined in the `gxComponentFlags` enumeration:

Constant	Value	Explanation
<code>gxOverResultComponent</code>	0x01	QuickDraw GX performs an AND operation between the result color and 0xFFFF before clamping.
<code>gxReverseComponent</code>	0x02	QuickDraw GX reverses the source and destination values before performing the transfer mode operation.

Specifying `gxOverResultComponent` allows the result of transfers using `gxAddMode` to wrap around (from 0xFFFF to 0x0000) instead of remaining clamped at 0xFFFF.

Specifying `gxReverseComponent` allows you to apply a transfer mode backwards—from the destination to the source—for a particular component. It is most useful for component modes that depend on order, like `gxMigrateMode`, or `gxAddMode` when used for subtraction.

Transfer Mode Flags

The transfer mode flags are a set of flags in the `flags` field of the transfer mode structure. They affect how color limits are used and whether a single component mode is to be used for all color components. There are three values for the flags, defined in the `gxTransferFlags` enumeration:

Constant	Value	Explanation
<code>gxRejectSourceTransfer</code>	0x0001	Negate the results of <code>sourceMinimum</code> and <code>sourceMaximum</code> for all components. Accept only values <i>outside</i> of the specified ranges.
<code>gxRejectDeviceTransfer</code>	0x0002	Negate the results of <code>deviceMinimum</code> and <code>deviceMaximum</code> for all components. Accept only values <i>outside</i> of the specified ranges.
<code>gxSingleComponentTransfer</code>	0x0004	Use a single transfer component for all color components. Duplicate <code>component[0]</code> in the transfer mode structure for all components in the transfer mode's color space.

Ink Objects

Setting the `gxRejectSourceTransfer` or `gxRejectDeviceTransfer` flag causes an inversion of the acceptable color ranges for source or destination color, respectively. For example, in Figure 5-14 on page 5-30, setting the `gxRejectSourceTransfer` or `gxRejectDeviceTransfer` flag would cause the white (empty) portions of the large cubes that represent RGB space to be within range, instead of the gray (filled) portions.

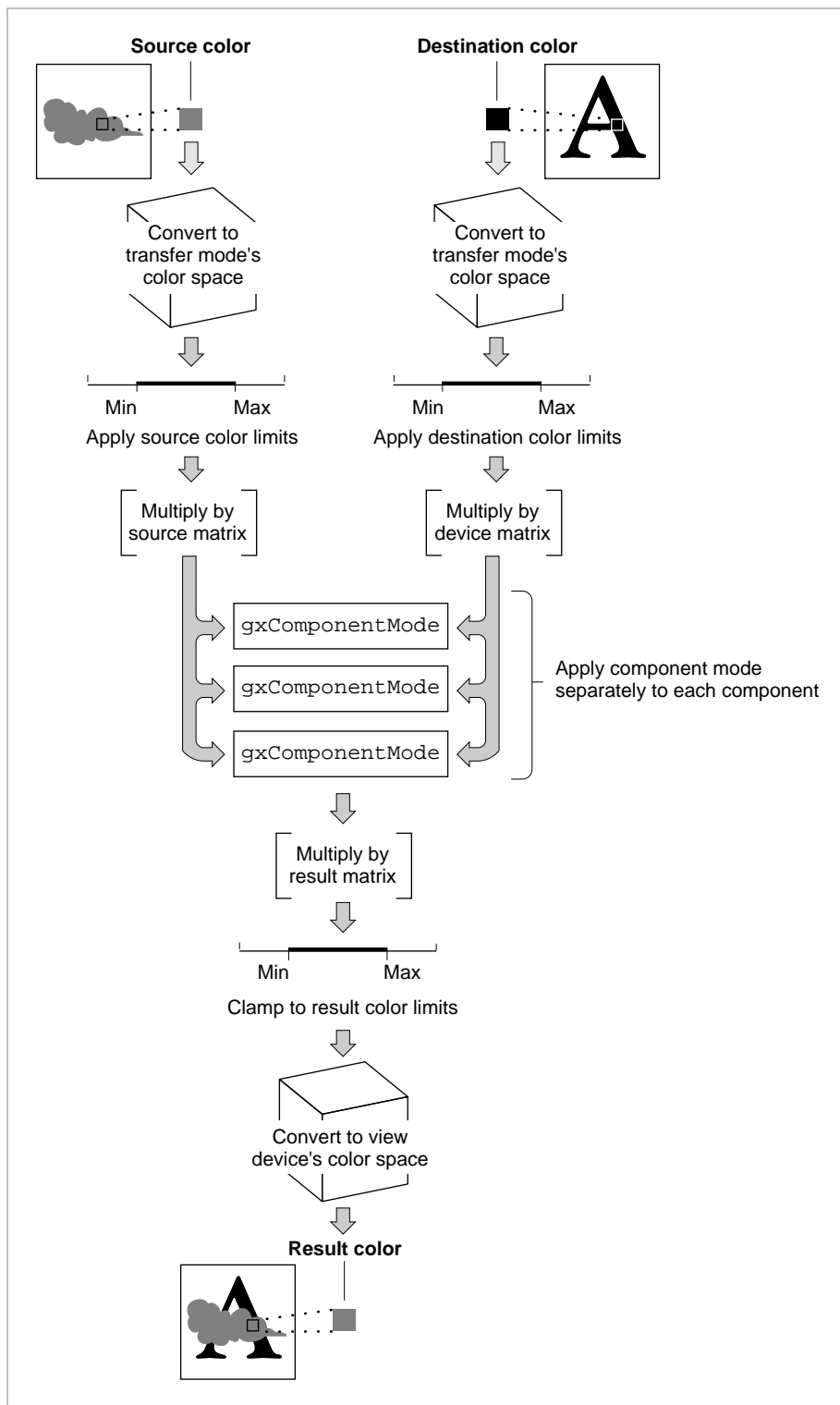
The effect is similar to, although not exactly the same as, individually reversing the minimum and maximum values for the color components. If the transfer mode flag is cleared, drawing occurs only when *all* components are inside the allowed ranges—that is, inside the darker gray portions of the large cubes in Figure 5-14. With the flag set, drawing occurs any time *at least one* component is outside of its allowed range—that is, with values anywhere outside of the dark gray areas in Figure 5-14.

The `gxSingleComponentTransfer` flag is provided as a convenience. You can set the flag when you don't need the flexibility (and extra effort) of specifying different transfer modes for different color components. In this case you need set up only one `gxTransferComponent` structure, instead of one for each component in the transfer mode's color space.

Summary of Transfer Mode Operation

Figure 5-18 shows how all the parts of the transfer mode structure work together when a color is drawn.

1. The source color is converted to the transfer mode's color space, if they are not already the same. Each component of the source color is then compared to the acceptable range of source colors, modified if appropriate by the values of the transfer mode flags. The resulting source components are then multiplied by the source matrix to yield the corrected source components for the transfer mode operation.
2. The destination color is converted to the transfer mode's color space, if necessary. Each component of the destination color is then compared to the acceptable range of destination colors, modified if appropriate by the values of the transfer mode flags. The resulting destination components are then multiplied by the device matrix to yield the corrected destination components for the transfer mode operation.
3. The pairs of source and destination components are each combined, according to the selected component mode, and the value of the operand if the component mode takes one. (Source and destination values for a component are swapped before combining if the `gxReverseComponent` component flag is set.)
4. The components that result from the transfer mode operation are each multiplied by the results matrix to yield the corrected result components (and then ANDed with `gxColorValue1` (0xFFFF) if the `gxOverResultComponent` flag is set for that component.) Each component is then pinned, if necessary, to the acceptable range of result colors. Finally, the result color is converted to the color space of the view device, and the color is drawn.

Figure 5-18 Summary of transfer mode operation

Using Ink Objects

This section describes how to create and use ink objects that support graphic or typographic shapes. This section describes how you can

- create and manipulate ink objects
- manipulate ink object properties
- get and set an ink's color
- work with transfer modes to achieve particular graphic effects

Creating and Manipulating Ink Objects

This section describes how you can create and interact with ink objects as whole entities—to create, dispose of, copy, compare, and clone them. Manipulating the individual properties of ink objects is described under “Manipulating Ink Object Properties” beginning on page 5-40.

Creating and Disposing of Ink Objects

QuickDraw GX provides the `GXNewInk` function to allow you to create a new ink object. You can also create a new ink object by copying an existing one with the `GXCopyToInk` function. Once you have created an ink object, you can customize its features using the techniques described in the section “Manipulating Ink Object Properties” beginning on page 5-40.

To delete your application's reference to an ink object, call the `GXDisposeInk` function, which may or may not actually release the memory allocated for that ink object, depending on the ink's owner count. The `GXDisposeInk` function decreases the ink object's owner count by 1; if that brings the owner count to zero, the ink is completely deleted and its memory released. See “Manipulating an Ink Object's Owner Count” beginning on page 5-41.

The following code fragment creates three ink objects in turn, gives each a color, assigns each to a shape (`windowShape1`, `windowShape2`, and `windowShape3`), adds each to a picture shape (`gOurHouse`), and then disposes of each ink reference because it is no longer needed. The code calls the library function `SetInkCommonColor` to assign colors to the individual ink objects.

```
gxInk redInk = GXNewInk();
SetInkCommonColor(redInk, red);
GXSetPictureParts(gOurHouse, 0, 0, 1, &windowShape1,
                  nil, &redInk, nil);
GXDisposeInk(redInk);
```

Ink Objects

```

gxInk grayInk = GXNewInk();
SetInkCommonColor(grayInk, gxGray);
GXSetPictureParts(gOurHouse, 0, 0, 1, &windowShape2,
                  nil, &grayInk, nil);
GXDisposeInk(grayInk);

gxInk turquoiseInk = GXNewInk();
SetInkCommonColor(turquoiseInk, turquoise);
GXSetPictureParts(gOurHouse, 0, 0, 1, &windowShape3,
                  nil, &turquoiseInk, nil);
GXDisposeInk(turquoiseInk);

```

The `GXNewInk` function is described on page 5-56. The `GXDisposeInk` function is described on page 5-57. The `GXSetPictureParts` function is described in the picture shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

Copying, Comparing, and Cloning Ink Objects

You can use the `GXCopyToInk` function to copy information from one ink object to another or to create a new copy of an ink object.

You can test if two ink-object references refer to the same ink object by simply testing the references for equality. You can also compare two different ink objects for equality with the `GXEqualInk` function. For two ink objects to be equal, their attributes, colors, color spaces, and transfer modes must have identical values, although their general object properties (owner count and tag list) do not need to be identical. Ink object copies created with the `GXCopyToInk` function are always equal to the ink from which they were copied.

The following code fragment effectively changes the default ink for a certain shape type. It makes a copy (`tempInk`) of the default ink object, modifies its color and transfer mode, and then assigns the modified copy to the default shape object for line shapes. After it makes the assignment, the code disposes of `tempInk`. The code makes use of the library functions `SetInkCommonColor` and `SetInkCommonTransfer` to set the ink's color and transfer mode.

```

tempInk = GXCopyToInk(nil,
                      GXGetShapeInk(GXGetDefaultShape(gxLineType)));
SetInkCommonColor(tempInk, gxBlack);
SetInkCommonTransfer(tempInk, xorMode);
GXSetShapeInk(GXGetDefaultShape(gxLineType), tempInk);
GXDisposeInk(tempInk);

```

In certain circumstances, you may want to copy a reference to an ink object without actually copying the ink object. For example, you may want two variables to refer to the same ink object, so that editing one of them affects both. This is called **cloning** an ink, rather than copying an ink. You can use the `GXCloneInk` function to clone an ink object.

Ink Objects

Functionally, `GXCloneInk` does nothing more than increase the owner count of an ink object. For more information about cloning objects, see the chapter “Introduction to Objects” in this book. For information on manipulating ink owner counts, see the section “Manipulating an Ink Object’s Owner Count” beginning on page 5-41 of this chapter.

The `GXCopyToInk` function is described on page 5-58. The `GXEqualInk` function is described on page 5-59. The `GXCloneInk` function is described on page 5-59.

Loading and Unloading Ink Objects

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

You call the `GXLoadInk` function to make sure that an ink object is in memory; if necessary, QuickDraw GX brings the object into memory from an unloaded state. You can call the `GXUnloadInk` function to instruct QuickDraw GX that it is free to unload the ink object at any time. These functions are described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Manipulating Ink Object Properties

This section describes how to manipulate the common object properties of ink objects: attributes, owner count, and tag list.

To manipulate the color of an ink, see the section “Getting and Setting an Ink Object’s Color” beginning on page 5-42. To manipulate the transfer mode of an ink, see the section “Getting and Setting an Ink Object’s Transfer Mode” beginning on page 5-43.

For manipulating an ink object as a whole, see “Creating and Manipulating Ink Objects” beginning on page 5-38.

Getting and Setting an Ink Object’s Attributes

You can use the `GXGetInkAttributes` function to find the attributes of an existing ink object, and the `GXSetInkAttributes` function to set the attributes of an ink. The following statements return the attributes for the source shape’s ink:

```
gxInkAttribute whatAttributes;
whatAttributes = GXGetInkAttributes(GXGetShapeInk(source));
```

As an example, to clear the `gxSuppressHalftoneInk` attribute of an ink referenced by the variable `target`, you could use the following statement:

```
GXSetInkAttributes(target,
    GXGetInkAttributes(target) & ~gxPortAlignDitherInk);
```

Ink Objects

Conversely, to set the `gxSuppressHalftoneInk` attribute, you could use the following line of code:

```
GXSetInkAttributes(target,
                    GXGetInkAttributes(target) | gxSuppressHalftoneInk);
```

Ink attributes are described in the section “Ink Attributes” beginning on page 5-9. The `GXGetInkAttributes` function is described on page 5-61. The `GXSetInkAttributes` function is described on page 5-62.

Manipulating an Ink Object’s Owner Count

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 ink object, QuickDraw GX sets the owner count of the new ink to 1. When you assign an existing ink object to a shape, QuickDraw GX increments the ink’s owner count, corresponding to the new reference to the ink contained in the shape object.

If you want to manage an ink’s owner count directly, or if you want to know whether an ink object is shared, you can use the `GXGetInkOwners` function to determine the owner count of an ink, and the `GXCloneInk` and `GXDisposeInk` functions to change the owner count of an ink. The `GXCloneInk` function increments the ink’s owner count, and the `GXDisposeInk` function decrements the ink’s owner count, freeing the memory used by the ink if the owner count goes to 0.

The `GXGetInkOwners` function is described on page 5-64.

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 ink and transform objects. Refer to that discussion if you find that ink objects you create have owner counts that are higher or lower than you expect.

Getting and Setting an Ink Object’s Tag References

You can examine the list of references to tag objects currently associated with an ink object using the `GXGetInkTags` function. Once you create a tag object, you can attach it to an ink object using the `GXSetInkTags` function. You can attach as many tag objects as you like to an ink object.

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 `GXGetInkTags` function is described on page 5-65. The `GXSetInkTags` function is described on page 5-66.

Ink Objects

Getting and Setting an Ink Object's Color

You can use the `GXGetInkColor` function to retrieve the color (as a `gxColor` structure) of an existing ink object; you can use the `GXGetShapeColor` function to retrieve the color of the ink object associated with a particular shape. You can use the `GXSetInkColor` function to assign a color to an ink; you can use the `GXSetShapeColor` function to assign a color to the ink object associated with a particular shape.

To simply get the color of an existing ink (referenced here by `myInk`) requires defining a color structure, then calling the `GXGetInkColor` function to fill it:

```
gxColor myInkColor;
GXGetInkColor(myInk, &myInkColor);
```

If you want to obtain some part of the ink's color structure, such as its color space, you could make calls like this:

```
gxColorSpace myInkSpace;
gxColor myInkColor;
myInkSpace = GXGetInkColor(myInk, &myInkColor)->space;
```

Conversely, to assign some portion of an ink's color structure, such as its color profile (here called `newProfile`), you could make these calls:

```
gxColor myInkColor;
GXGetInkColor(myInk, &myInkColor);
myInkColor.profile = newProfile;
GXSetInkColor(myInk, &myInkColor);
```

You can give a shape a specific color by setting the individual values of its ink's color components. For a shape (`myShape`) whose ink uses an RGB color space, you could do something like this (with numeric color values `newRed`, `newBlue`, and `newGreen`):

```
gxColor newColor;
newColor.space = gxRGBSpace;
newColor.profile = nil;
newColor.element.rgb.red = newRed;
newColor.element.rgb.green = newGreen;
newColor.element.rgb.blue = newBlue;
GXSetShapeColor(myShape, &newColor);
```

The `GXGetInkColor` function is described on page 5-68; the `GXGetShapeColor` function is described on page 5-70. The `GXSetInkColor` function is described on page 5-69; the `GXSetShapeColor` function is described on page 5-71.

Colors and how to program with them are not described further in this chapter; for complete information, see the chapter "Colors and Color-Related Objects" in this book.

Getting and Setting an Ink Object's Transfer Mode

You can use the `GXGetInkTransfer` function to retrieve the transfer mode (as a `gxTransferMode` structure) of an existing ink object; you can use the `GXGetShapeTransfer` function to retrieve the transfer mode of the ink object associated with a particular shape. You can use the `GXSetInkTransfer` function to assign a transfer mode to an ink; you can use the `GXSetShapeTransfer` function to assign a transfer mode to the ink object associated with a particular shape.

To simply get the transfer mode of an existing ink (referenced here by `myInk`) requires defining a transfer mode structure, then calling `GXGetInkTransfer` to fill it:

```
gxTransferMode myInkTransfer;
GXGetInkTransfer(myInk, &myInkTransfer);
```

If you want to obtain some part of the ink's transfer mode structure, such as its color space, you could make calls like this:

```
gxTransferMode myInkTransfer;
gxColorSpace myTransferSpace;
myTransferSpace = GXGetInkTransfer(myInk, &myInkTransfer)->space;
```

Conversely, to set some portion of an ink's transfer mode structure, such as one of its transfer component structures (in an array here called `newComponents`), you could make these calls:

```
gxTransferMode myInkTransfer;
GXGetInkTransfer(myInk, &myInkTransfer);
myInkTransfer.component[2] = newComponents[2];
GXSetInkTransfer(myInk, &myInkTransfer);
```

You can alter a shape's transfer mode type by setting the component mode for one component of its ink's transfer mode. For example, you can change the transfer mode type of the first color component of the shape `myShape` to `gxMinimumMode` with calls like this:

```
gxTransferMode myShapeTransfer;
GXGetShapeTransfer(myShape &myShapeTransfer);
myShapeTransfer.component[0].mode = gxMinimumMode;
GXSetShapeTransfer(myShape, &myShapeTransfer);
```

The `GXGetInkTransfer` function is described on page 5-72; the `GXGetShapeTransfer` function is described on page 5-74. The `GXSetInkTransfer` function is described on page 5-73; the `GXSetShapeTransfer` function is described on page 5-75.

Transfer modes and the transfer mode structure are described in the section "About Transfer Modes" beginning on page 5-11. The next section describes how to use transfer modes to get particular drawing effects.

Working With Transfer Modes

This section describes some of the ways to use transfer modes for drawing. Note that there are many ways to use transfer modes in drawing; this section mentions only a few of the more common possibilities.

See “About Transfer Modes” beginning on page 5-11 for a discussion of the complex process by which QuickDraw GX performs transfer mode calculations.

Simple Source-to-Destination Transfers

To simply draw a color, shape, pattern, or bitmap to the destination, regardless of what the destination currently contains, use `gxCopyMode` for all color components. Use identity matrices and make sure all your source colors are within any color limits you are using. Clear all flags (except `gxSingleComponentTransfer`, if you are using it to make sure all your components use `gxCopyMode` with the same color limits and component flags). If your application is drawing or painting with a single color, `gxCopyMode` means that the color is applied without modification to all parts of the destination you are drawing to.

Copy mode is especially fast and efficient for drawing because the characteristics of the destination are not taken into account.

You can also use `gxAddMode`, `gxBlendMode`, or `gxMigrateMode` to draw the entire source, but in ways that combine the source and destination. If your application is drawing or painting with a single source color, these modes cause the drawn color or colors to be some combination of the source and the destination. For example, drawing a red apple onto a blue background, using `gxAddMode` in RGB space, results in a magenta apple against a blue background.

You can use `gxBlendMode` mode, for example, to lighten or darken all shapes in a picture by some ratio compared to the background (destination). The following code fragment sets the transfer mode of a shape (`theShape`) so that the shape’s color is blended in a given percentage (`thePercentage`, normalized to `gxColorValue1`) with the destination color.

```
gxTransferMode    shapeTransfer;
GXGetShapeTransfer(theShape, &shapeTransfer);

/* use single-component transfer, blend mode */
shapeTransfer.flags = gxSingleComponentTransfer;
shapeTransfer.component[0].mode = gxBlendMode;

shapeTransfer.component[0].flags = 0;
shapeTransfer.component[0].sourceMinimum = 0;
shapeTransfer.component[0].sourceMaximum = gxColorValue1;
shapeTransfer.component[0].deviceMinimum = 0;
shapeTransfer.component[0].deviceMaximum = gxColorValue1;
```


Ink Objects

```

shapeTransfer.component[0].clampMinimum = 0;
shapeTransfer.component[0].clampMaximum = gxColorValue1;
shapeTransfer.component[0].operand =
    ((unsigned long)gxColorValue1) * thePercentage)/100;

GXSetShapeTransfer(theShape, &shapeTransfer);

```

Drawing Selected Parts of the Source

There are many ways to transfer only parts of the source image to the destination.

You can use `gxMinimumMode` to transfer only those parts of the source that are darker than the destination; if your application is drawing or painting with a single source color, `gxMinimumMode` has the effect of darkening and coloring the lighter parts of the destination.

You can use `gxMaximumMode` to transfer only those parts of the source that are lighter than the destination; if your application is drawing or painting with a single source color, `gxMaximumMode` has the effect of lightening and coloring the darker parts of the destination.

You can use `gxCopyMode` but set source color limits so that only colors within certain ranges are transferred. If, for example, part of your source image is bright red, you can set a maximum limit on red intensity in the source; drawing will not occur where that bright red exists, and your destination image will “show through” in those places. You can work in HSV space and set limits on source luminance, so that, for example, your destination image will “show through” the highlights or the shadows of your source image after drawing.

If you are drawing in black and white, you can use `gxAndMode` or `gxRampAndMode` to transfer only those white parts of the source that are identical to the destination. Alternatively, you can use `gxXorMode` or `gxRampXorMode` to transfer only those white parts of the source that are different from the destination. The modes `gxOrMode` and `gxRampOrMode` transfer all of the white parts of the source to the destination. (For colors other than black and white, Boolean modes give unpredictable results, and pseudo-Boolean modes give results that look like blended versions of black-and-white Boolean.)

If you want to mask off parts of the source image that cannot be defined simply, in terms of color or intensity, you can use alpha-channel modes. See the section “Masking” on page 5-48.

Preserving Selected Parts of the Destination

Preserving parts of the destination image is equivalent to drawing only parts of the source, except that it is the characteristics of the destination, not the source, that determine where drawing does and does not occur.

Ink Objects

The modes `gxMinimumMode` and `gxMaximumMode` base drawing on destination characteristics as well as on source characteristics, so you can pick mode and source colors to make sure that desired parts of the destination remain unchanged after drawing. For example, if your destination has bright blue letters on a black background, you can replace that background by drawing with `gxMaximumMode` and using any color or image darker than the letters. If you are working in HSV space, you could, for example, turn the background to a different color (of any intensity) by drawing with `gxMaximumMode`, and using a color (such as yellow) whose hue value is less than that of the blue letters. You could even leave the background black and change the color of the letters by specifying `gxMinimumMode` for the saturation component and using any source color less saturated than the blue of the letters.

You can use `gxCopyMode` but set destination color limits so that drawing occurs only where the destination colors are within certain ranges. If, for example, the black parts of your destination image must be preserved, you can set a minimum limit on the luminance of the destination; drawing will not occur where that black exists, letting the black parts of the destination “show through” the source image after drawing. If you want to preserve only the summer sky in a destination image, set the destination color limits to block drawing in the blue range.

If you are drawing in black and white, you can use `gxAndMode` or `gxRampAndMode` to preserve only those white parts of the destination that are identical to the source. Alternatively, you can use `gxXorMode` or `gxRampXorMode` to preserve only those white parts of the destination that are different from the source (and to turn black any white parts of the destination that are also white in the source). The modes `gxOrMode` and `gxRampOrMode` preserve all of the white parts of the destination, adding to them all of the white parts of the source. (For colors other than black and white, Boolean modes give unpredictable results, and pseudo-Boolean modes give results that look like blended versions of black-and-white Boolean.)

Copying or Preserving Luminance

In some cases you may want to alter the colors but not the lightness of an image, or you may want to apply a color, pattern, or texture to an object in a grayscale image. What you are doing is preserving the luminance in either the source or destination, while letting other color components vary when you draw.

If, for example, your source image is a picture of a teapot, and your destination image is a color or pattern, you can color the teapot by drawing (in HSV space) with `gxNoMode` in the hue and saturation components, and `gxCopyMode` in the value (intensity) component.

Alternatively, if the color to apply is in the source image and the teapot is in the destination image, you could use `gxCopyMode` in the hue and saturation components, and `gxNoMode` in the value component. (Or you could draw as described in the previous paragraph, but set the `gxReverseComponent` flag in each component before drawing, although that is a less efficient way to draw.) Figure 5-19 shows an example of drawing by preserving destination luminance in that way. Color Plate 3 at the front of this book shows the same drawing sequence in color.

Figure 5-19 Applying color by preserving luminance in the destination

If your application is drawing or painting with a single source color, you can apply that color to the destination image—making it a single hue, but not otherwise changing the destination—by applying `gxCopyMode` to the hue component, and `gxNoMode` to the other components, including luminance.

Modifying Luminance

Modifying luminance can be used to make a dark image lighter or to make a light image darker, to increase its brightness range (contrast), or to create an overlay or blend of one image onto another, without affecting the hue or saturation of the destination. You can change the luminance of a destination image in several ways, most easily from within HSV or HLS color space.

You can draw to the destination using a source image that has any hue or saturation but the desired luminance, using `gxCopyMode` for luminance and `gxNoMode` for the other components. (That's the same as preserving the luminance of the source, as described in the previous section.)

You can draw to the destination with a source image whose luminance is equal to (or some multiple of, or some absolute amount above or below) the destination image, and use `gxAddMode` or `gxCopyMode`. The effect is to uniformly increase or decrease the luminance of the destination image. (Negative luminances are not permissible inputs to transfer mode calculations, although you can achieve the same effect by multiplying luminance by -1 in the transfer mode matrices.) The same effect can be achieved, however, by drawing with `gxNoMode` for all components and then applying the result matrix to add to the luminance or multiply it by some factor.

Isolating and Modifying Color Ranges

You can restrict drawing to individual colors or ranges of colors in a number of ways. For example, in RGB space you can draw mostly the reds by specifying very restrictive low source color limits for the other components, although this method wouldn't prevent dark non-red colors from being drawn. To draw mostly yellows, you could convert to CMYK space and do the same.

Ink Objects

In HSV space, you can set the source (or destination) color limits on the hue component, in order to draw (or preserve from drawing) an exact range of hues, independent of their brightness or saturation. You could, for instance, isolate greens in a source landscape image in order to draw fields and trees but not sky; or you could isolate flesh tones in a destination portrait image in order to change hair and clothing colors but not skin color.

You can change the colors in the range you have isolated by using the source, device, or result matrices to shift the hue component. Thus you could change green fields to shades of tan and brown, or even change tan and brown skin to shades of green.

Masking

You can draw an irregular portion of a source image over a destination image, letting the destination show around the edges of the source portion and through holes in it, in several ways. In simple situations, selecting the right transfer mode can accomplish what you need, as discussed under “Drawing Selected Parts of the Source” on page 5-45 and “Preserving Selected Parts of the Destination” on page 5-45.

Drawing parts of a complex image on a complex background usually requires the use of a mask image that controls what parts of the source get drawn and what parts do not. QuickDraw GX allows you to integrate a mask with any source or destination image by storing transparency information in the alpha channel of each pixel’s color.

You can make parts of your source image transparent and other parts opaque by placing either 0 or `gxColorValue1`, respectively, in the alpha-channel component of each pixel’s color. Then, when you draw your source image to the destination, use one of the alpha-channel transfer modes to get the masking effect you want. The mode `gxOverMode` is probably the most common transfer mode you’ll use; it’s like `gxCopyMode` with transparency added.

The destination image may have transparency also, and if you want to use the result image as a source image for subsequent drawing, you need to calculate the result alpha-channel values as well. With `gxOverMode` for the color components, the most likely mode to use on the alpha channel itself is `gxRampOrMode`.

Partial Transparency

Besides masking, you can use alpha-channel values to draw images transparently over other images, to give a translucent “stained-glass” or ghostly effect, or to draw a pattern or texture as a transparent surface effect or shadow on an image.

In simple situations, just using `gxBlendMode` or `gxMigrateMode` might give the translucent effect you need; the intersection of a white rectangle and a black rectangle is gray, which might be enough. For more sophisticated effects, you can define alpha-channel values in your source image that let the destination image show through partially or completely, and then pick an alpha-channel transfer mode that is appropriate, given the transparency of both source and destination and the transparency you want the result to retain.

Anti-Aliasing

Alpha-channel values and alpha-channel modes are also used for anti-aliasing, a drawing technique that minimizes jagged edges around objects drawn on the screen. Type at large sizes is typically drawn with anti-aliasing to smooth its appearance.

For anti-aliasing, you first set alpha-channel values to create a mask that defines the opaque parts of your image. Then, at the edges of the opaque portions, you define a zone of partial transparency, one or more pixels wide, that creates a transition from opacity within to transparency without. The color and transparency of each pixel are commonly computed based on the portion of the pixel that the opaque object is calculated to cover. Figure 5-9 on page 5-25 shows an example of this effect.

When you then draw the object with an alpha-channel transfer mode, its edges are feathered, with colors transitional between the source and destination colors. Diagonal lines and curves thus appear smoother and less jagged.

Making Color Separations

Creating color separations from images is fundamentally straightforward with QuickDraw GX. For CMYK color separations, you can use a transfer mode structure that works with CMYK color space and draw your image four times, each time using `gxCopyMode` for all components but modifying the source matrix each time to pass through only the component that you want.

You can also create halftones for each separation. See the chapter “View-Related Devices” in this book for a discussion of halftones in QuickDraw GX.

Transfer Modes and Printing

Printers are imaging devices, and the printing components of QuickDraw GX support all transfer modes. When an image is printed, the original state of the destination (paper) is treated as if it were a white image—equivalent to `0xFFFF` or `gxColorValue1` in all components of RGB space, for example. QuickDraw GX then accumulates all drawing commands before actually printing the page. As it draws each shape on the page, QuickDraw GX combines the source image with the destination, which may be white or may reflect the results of previous drawing, according to the transfer mode selected. After the last shape is drawn, QuickDraw GX prints the result. Thus all transfer modes, and even alpha-channel values, can be accounted for in printing.

Even vector devices such as plotters can support transfer modes. The color of each shape the plotter is to draw is combined with the destination according to whatever transfer mode is selected, and at the end the resulting color is printed using one of the available pen colors and patterns.

Ink Objects

Printing in QuickDraw GX is optimized for certain transfer mode configurations. In general, printing is fastest using the default ink object's transfer mode (`gxCopyMode` for all components, identity matrices, wide-open color limits). In addition, for fastest printing of 1-bit-per-pixel bitmaps, QuickDraw GX recognizes a special configuration that replicates the QuickDraw `srcOr` transfer mode on the Macintosh: `gxCopyMode` in all components and source color limits set so that only the “on” or “foreground” bits of the image are printed.

For more information on printing for applications, see *Inside Macintosh: QuickDraw GX Printing*. For more information on printing for printer drivers and extensions, see *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

Ink Objects Reference

This section provides reference information about the data structures and functions that allow you to create and manipulate ink objects, and to work with transfer modes. It includes

- descriptions of the constants and data types used by ink objects
- descriptions of the QuickDraw GX functions that operate on ink objects, including those that manipulate ink color and transfer mode

Constants and Data Types

This section describes the constants and data types that define

- ink attributes
- the color structure
- transfer modes

The documentation for transfer modes is complete in this section; the documentation for colors is summary only. Additional reference information on colors is found in the chapter “Colors and Color-Related Objects” in this book.

The Ink Object

QuickDraw GX provides you with access to an individual ink object through an ink reference:

```
typedef struct gxPrivateInkRecord *gxInk;
```

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

Ink Attributes

Each ink object has a set of ink attributes, defined in the `gxInkAttributes` enumeration. The attributes specify how to handle dithering and halftoning.

```
enum gxInkAttributes{
    gxPortAlignDitherInk = 0x0001,
    gxForceDitherInk     = 0x0002,
    gxSuppressDitherInk  = 0x0004,
    gxSuppressHalftoneInk= 0x0008
};

typedef long gxInkAttribute;
```

The individual ink attributes are described in the section “Ink Attributes” beginning on page 5-9.

Color Structure

The color property of an ink object is specified with a color structure (type `gxColor`):

```
struct gxColor{
    gxColorSpace space;
    gxColorProfile profile;
    union {
        struct gxCMYKColor    cmyk;
        struct gxRGBColor     rgb;
        struct gxRGBAColor    rgba;
        struct gxHSVColor     hsv;
        struct gxHLSColor     hls;
        struct gxCIEColor     cie;
        struct gxYIQColor     yiq;
        gxColorValue          gray;
        struct gxGrayaColor    graya;
        unsigned short        pixel16;
        unsigned long         pixel32;
        struct gxIndexedColor  indexed;
        gxColorValue          component[4];
    } element;
};
```

The fields of the color structure are described briefly in the section “Color” beginning on page 5-7 of this chapter, and defined in more detail in the chapter “Colors and Color-Related Objects” in this book.

Transfer Mode Structure

The transfer mode structure (data type `gxTransferMode`) specifies the transfer mode property of an ink object.

```
struct gxTransferMode{
    gxColorSpace          space;
    gxColorSet            set;
    gxColorProfile        profile;
    Fixed                 sourceMatrix[5][4];
    Fixed                 deviceMatrix[5][4];
    Fixed                 resultMatrix[5][4];
    gxTransferFlag        flags;
    struct gxTransferComponent component[4];
};
```

Field descriptions

<code>space</code>	The color space used by this transfer mode. The color spaces defined by QuickDraw GX are listed in the color structure definition shown in the previous section, and are described in the chapter “Colors and Color-Related Objects” in this book. A transfer mode’s color space need not be the same as the color space of the ink object the transfer mode is part of.
<code>set</code>	A reference to a color set, an object that defines a list of colors available for use by this transfer mode. This field has meaning only if the transfer mode’s color space is <code>gxIndexedSpace</code> ; if the value in this field is <code>nil</code> , there is no color set associated with the transfer mode. Color sets are described in the chapter “Colors and Color-Related Objects” in this book.
<code>profile</code>	A reference to a color profile, an object that specifies color-correction information. If the value in this field is <code>nil</code> , there is no color profile associated with the color space of this transfer mode. Color profiles are described in the chapter “Colors and Color-Related Objects” in this book.
<code>sourceMatrix</code>	A 5 x 4 matrix that specifies how to transform the source color before applying the component modes to the components. See “Transfer Mode Matrices” beginning on page 5-33.
<code>deviceMatrix</code>	A 5 x 4 matrix that specifies how to transform the destination color before applying the component modes to the components. See “Transfer Mode Matrices” beginning on page 5-33.
<code>resultMatrix</code>	A 5 x 4 matrix that specifies how to transform the result color after applying the component modes to all components. See “Transfer Mode Matrices” beginning on page 5-33.
<code>flags</code>	The transfer mode flags; they specify how to handle color limits and whether multiple transfer components are needed. The transfer mode flags are described in the section “Transfer Mode Flags” on page 5-35.

Ink Objects

component An array of four transfer component structures, each specifying the type of transfer mode to apply to a single color component of the transfer mode's color space. The transfer component structure is described next.

Transfer Mode Flags

The transfer mode flags are defined in the `gxTransferFlags` enumeration:

```
enum gxTransferFlags{
    gxRejectSourceTransfer    = 0x0001,    /* ≥ 1 source component
                                           must be out of range */
    gxRejectDeviceTransfer    = 0x0002,    /* ≥ 1 device component
                                           must be out of range */
    gxSingleComponentTransfer= 0x0004    /* transfer component[0]
                                           = all components */
};

typedef long gxTransferFlag;
```

The individual transfer mode flags are described in the section “Transfer Mode Flags” on page 5-35.

Transfer Component Structure

There are up to four transfer component structures in a transfer mode structure. Each transfer component structure describes how the transfer mode operation is to be applied to a given component in the transfer mode's color space. The transfer component structure is of data type `gxTransferComponent`:

```
struct gxTransferComponent{
    gxComponentMode    mode;
    gxComponentFlag    flags;
    gxColorValue        sourceMinimum;
    gxColorValue        sourceMaximum;
    gxColorValue        deviceMinimum;
    gxColorValue        deviceMaximum;
    gxColorValue        clampMinimum;
    gxColorValue        clampMaximum;
    gxColorValue        operand;
};
```

Ink Objects

Field descriptions

mode	The component mode (type of transfer mode, such as <code>gxCopyMode</code> or <code>gxBlendMode</code>) to apply to this color component. Component modes are described in the section “Transfer Mode Types” beginning on page 5-11.
flags	The component flags, which control clamping behavior and whether source and destination are to be reversed for this component. The component flags are described in the section “Transfer Component Flags” on page 5-35.
sourceMinimum	The minimum acceptable value for source color in this color component. No drawing occurs if the source component value is below <code>sourceMinimum</code> . For more information, see “Color Limits” beginning on page 5-27, and “Source Color Limits” on page 5-31.
sourceMaximum	The maximum acceptable value for source color in this color component. No drawing occurs if the source component value is greater than <code>sourceMaximum</code> . For more information, see “Color Limits” beginning on page 5-27, and “Source Color Limits” on page 5-31.
deviceMinimum	The minimum acceptable value for destination color in this color component. No drawing occurs if the destination component value is below <code>deviceMinimum</code> . For more information, see “Color Limits” beginning on page 5-27, and “Destination Color Limits” on page 5-32.
deviceMaximum	The maximum acceptable value for destination color in this color component. No drawing occurs if the destination component value is greater than <code>deviceMaximum</code> . For more information, see “Color Limits” beginning on page 5-27, and “Destination Color Limits” on page 5-32.
clampMinimum	The minimum acceptable value for result color in this color component. If the result component value is below <code>deviceMinimum</code> , it is pinned, or clamped, to the value of <code>deviceMinimum</code> . For more information, see “Color Limits” beginning on page 5-27, and “Result Color Limits” on page 5-32.
clampMaximum	The maximum acceptable value for result color in this color component. If the result component value is greater than <code>deviceMaximum</code> , it is pinned, or clamped, to the value of <code>deviceMaximum</code> . For more information, see “Color Limits” beginning on page 5-27, and “Result Color Limits” on page 5-32.
operand	A value used as an input to the <code>gxBlendMode</code> , <code>gxMigrateMode</code> , and <code>gxHighlightMode</code> component modes. If you are using these modes, you must supply a proper value for operand. For more information, see “Arithmetic Transfer Modes” beginning on page 5-12, and “Highlight Transfer Mode” on page 5-15.

Component Modes (Transfer Mode Types)

QuickDraw GX supports the following types of transfer modes, defined in the `gxComponentModes` enumeration:

```
enum gxComponentModes{
    gxNoMode = 0,
    gxCopyMode,
    gxAddMode,
    gxBlendMode,
    gxMigrateMode,
    gxMinimumMode,
    gxMaximumMode,
    gxHighlightMode,
    gxAndMode,
    gxOrMode,
    gxXorMode,
    gxRampAndMode,
    gxRampOrMode,
    gxRampXorMode,
    gxOverMode,
    gxAtopMode,
    gxExcludeMode,
    gxFadeMode
};

typedef unsigned char gxComponentMode;
```

The individual component modes are described in the section “Transfer Mode Types” beginning on page 5-11.

Transfer Component Flags

The transfer component flags are part of the transfer component structure (data type `gxTransferComponent`). The flags are defined in the `gxComponentFlags` enumeration:

```
enum gxComponentFlags{
    gxOverResultComponent= 0x01, /* AND result component with
                                   0xFFFF before clamping */
    gxReverseComponent    = 0x02 /* Reverse source and device
                                   components before mode */
};

typedef unsigned char gxComponentFlag;
```

Ink Objects

The individual transfer component flags are described in the section “Transfer Component Flags” on page 5-35.

Functions

This section describes the QuickDraw GX functions you can use to

- create and manipulate an ink object
- manipulate the common object properties of an ink object
- get and set the color of an ink object
- get and set the transfer mode of an ink object

Creating and Manipulating Ink Objects

This section describes the functions that manipulate inks as objects in memory. With the functions in this section, you can create and dispose of an ink object, and copy, compare, and clone ink objects.

To associate an ink object with a QuickDraw GX shape object, use the `GXGetShapeInk` and `GXSetShapeInk` functions, described in the chapter “Shape Objects” in this book.

GXNewInk

You can use the `GXNewInk` function to create a new ink object with default properties.

```
gxInk GXNewInk(void);
```

function result A reference to a newly created copy of the default ink object.

DESCRIPTION

The `GXNewInk` function creates an ink object with an owner count of 1. All other properties of the ink are set to their default values:

- No attributes set.
- An empty tag list.
- An owner count of 1.
- Color space set to `gxRGBSpace` with each color component set to 0, which represents black in this color space.
- Transfer mode set to `gxCopyMode`, with identity transfer mode matrices, color limits of 0 to 0xFFFF (`gxColorValue1`), and all flags cleared.

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewInk` function creates an ink object; you are responsible for disposing of that object when you no longer need it.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`

SEE ALSO

Default ink values are described in the section “The Default Ink Object” on page 5-10.

GXDisposeInk

You can use the `GXDisposeInk` function to release a reference to an ink object.

```
void GXDisposeInk(gxInk target);
```

`target` A reference to the ink object to dispose of.

DESCRIPTION

The `GXDisposeInk` function decrements the owner count of the ink object specified by the `target` parameter and releases any memory used by it if the owner count goes to 0.

ERRORS, WARNINGS, AND NOTICES**Errors**

`ink_is_nil`

SEE ALSO

Owner counts for ink objects are discussed in the section “Copying, Comparing, and Cloning Ink Objects” beginning on page 5-39, and in the section “Manipulating an Ink Object’s Owner Count” beginning on page 5-41. To examine the owner count of an ink, use the `GXGetInkOwners` function, described on page 5-64.

GXCopyToInk

You can use the `GXCopyToInk` function to create a copy of an existing ink object.

```
gxInk GXCopyToInk(gxInk target, gxInk source);
```

target A reference to the ink object to copy the `source` contents into. If you specify `nil` for this parameter, the `GXCopyToInk` function creates a new ink object.

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

function result A reference to the copy (that is, the target ink).

DESCRIPTION

The `GXCopyToInk` function copies the contents of an existing ink object to another, or it creates a new ink object and copies the contents of an existing ink object to it. The function copies the color, transfer mode, attributes, and tag list (but not the owner count) of the ink object specified by the `source` parameter into the ink 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, the `GXCopyToInk` function creates a new ink object and copies the source properties, including owner count and tag list, into it.

You can use the `GXCopyToInk` function to create a copy of an ink object so you can modify it without changing the original.

SPECIAL CONSIDERATIONS

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

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`ink_is_nil`

SEE ALSO

To create a new ink that is a copy of the default ink instead of a copy of an existing ink, use the `GXNewInk` function, described on page 5-56.

To compare two ink objects, use the `GXEqualInk` function, described in the next section.

GXEqualInk

You can use the `GXEqualInk` function to determine whether two ink objects are equal.

```
boolean GXEqualInk(gxInk one, gxInk two);
```

`one` A reference to one of the ink objects to test for equality.

`two` A reference to the other ink object to test for equality.

function result `true` if the ink specified by the `one` parameter is equal to the ink specified by the `two` parameter; otherwise `false`.

DESCRIPTION

The `GXEqualInk` function returns as its function result a Boolean value indicating whether the ink object specified by the `one` parameter is equal to the ink object specified by the `two` parameter.

For two ink objects to be equal, they must have identical colors, transfer modes, and attributes, although their owner count and tag list need not be identical.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`ink_is_nil`

SEE ALSO

To make a copy of an ink object that is equal by the criteria of this function, use the `GXCopyToInk` function, described in the previous section.

GXCloneInk

You can use the `GXCloneInk` function to clone an ink object—that is, to add a reference to it and increment its owner count.

```
gxInk GXCloneInk(gxInk source);
```

`source` A reference to the ink object to clone.

function result A reference to the cloned ink.

Ink Objects

DESCRIPTION

The `GXCloneInk` function increments the owner count of the ink referenced in the `source` parameter. You typically use this function when you want to create another reference to an existing ink instead of creating a distinct copy of the ink.

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

ERRORS, WARNINGS, AND NOTICES**Errors**

`ink_is_nil`

SEE ALSO

Owner counts for ink objects are discussed in the section “Copying, Comparing, and Cloning Ink Objects” beginning on page 5-39, and in the section “Manipulating an Ink Object’s Owner Count” beginning on page 5-41.

To examine the owner count of an ink, use the `GXGetInkOwners` function, described on page 5-64. To decrement the owner count of an ink, use the `GXDisposeInk` function, described on page 5-57.

Manipulating Ink Object Properties

The functions described in this section allow you to

- reset an ink’s properties to their default values
- manipulate the common object properties of an ink object: attributes, owner count, and tag list

GXResetInk

You can use the `GXResetInk` function to reset the properties of an ink to their default values.

```
void GXResetInk(gxInk target);
```

`target` A reference to the ink object whose properties you want to reset.

DESCRIPTION

The `GXResetInk` function resets the color, transfer mode, and attributes of the ink object specified by the `target` parameter to match the properties of the default ink object:

- No attributes set.

Ink Objects

- Color space set to `gxRGBSpace` with each color component set to 0, which represents black in this color space.
- Transfer mode set to `gxCopyMode`, with identity transfer mode matrices, color limits of 0 to 0xFFFF (`gxColorValue1`), and all flags cleared.

The `GXResetInk` function does not change the target ink's owner count or tag list.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`ink_is_nil`

SEE ALSO

Default ink properties are described in the section “The Default Ink Object” on page 5-10.

GXGetInkAttributes

You can use the `GXGetInkAttributes` function to examine which attributes of an ink object are set.

```
gxInkAttribute GXGetInkAttributes(gxInk source);
```

`source` A reference to the ink to find the attributes of.

function result The ink attributes of the source ink object.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`ink_is_nil`

SEE ALSO

Ink attributes are described in the section “Ink Attributes” beginning on page 5-9.

To change the attributes of an ink object, use the `GXSetInkAttributes` function, described in the next section.

To examine the attributes of the ink object associated with a specified shape, use the `GXGetShapeInkAttributes` function, described on page 5-62.

GXSetInkAttributes

You can use the `GXSetInkAttributes` function to set or clear the attributes of an ink object.

```
void GXSetInkAttributes(gxInk target, gxInkAttribute attributes);
```

`target` A reference to the ink object to change the attributes of.

`attributes` The new ink attributes to be assigned.

DESCRIPTION

The `GXSetInkAttributes` function sets the attributes of the ink object referenced in the `target` parameter to those specified in the `attributes` parameter. If you pass `gxNoAttributes` for the `attributes` parameter, all attributes are cleared.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`ink_is_nil`

`parameter_out_of_range` (debugging version)

Notices (debugging version)

`attributes_already_set`

SEE ALSO

Ink attributes are described in the section “Ink Attributes” beginning on page 5-9.

To examine the attributes of an ink object, use the `GXGetInkAttributes` function, described in the previous section.

To change the attributes of the ink object associated with a specified shape, use the `GXSetShapeInkAttributes` function, described on page 5-63.

GXGetShapeInkAttributes

You can use the `GXGetShapeInkAttributes` function to examine the attributes of the ink associated with a shape.

```
gxInkAttribute GXGetShapeInkAttributes(gxShape source);
```

`source` A reference to the shape whose ink object you want the attributes of.

function result The attributes of the ink object associated with the source shape object.

ERRORS, WARNINGS, AND NOTICES**Errors**

out_of_memory
shape_is_nil

SEE ALSO

To set the attributes of the ink object associated with a shape, use the `GXSetShapeInkAttributes` function, described next.

To examine the attributes of an ink object itself, use the `GXGetInkAttributes` function, described on page 5-61.

Ink attributes are described in the section “Ink Attributes” beginning on page 5-9.

GXSetShapeInkAttributes

You can use the `GXSetShapeInkAttributes` function to set or clear attributes of the ink associated with a shape.

```
void GXSetShapeInkAttributes(gxShape target,
                             gxInkAttribute attributes);
```

target A reference to the shape whose ink object you want to change the attributes of.

attributes The new ink attributes to be assigned.

DESCRIPTION

The `GXSetShapeInkAttributes` function assigns the ink attributes specified by the `attributes` parameter to the ink object associated with the shape referenced in the `target` parameter. It is almost equivalent to the following call:

```
GXSetInkAttributes(GXGetShapeInk(target), attributes);
```

The only difference is that, if the source shape’s ink object is shared with other shapes, `GXSetShapeInkAttributes` creates a new copy of the ink object, attaches it to the source shape, and changes the attributes of the copy. That way, calling this function does not produce side effects on other shapes.

Ink Objects

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory

shape_is_nil

parameter_out_of_range (debugging version)

Notices (debugging version)

attributes_already_set

SEE ALSO

To examine the attributes of the ink object associated with a shape, use the `GXGetShapeInkAttributes` function, described in the previous section.

To set the attributes of an ink object itself, use the `GXSetInkAttributes` function, described on page 5-62.

Ink attributes are described in the section “Ink Attributes” beginning on page 5-9. The `GXSetInkAttributes` function is described on page 5-62.

The `GXGetShapeInk` function is described in the chapter “Shape Objects” in this book.

GXGetInkOwners

You can use the `GXGetInkOwners` function to determine the number of references to a particular ink object.

```
long GXGetInkOwners(gxInk source);
```

`source` A reference to the ink to find the owner count of.

function result The owner count of the source ink object.

DESCRIPTION

The `GXGetInkOwners` function returns as its function result the current number of references to the ink object.

ERRORS, WARNINGS, AND NOTICES

Errors

ink_is_nil

SEE ALSO

Owner counts for ink objects are discussed in the section “Copying, Comparing, and Cloning Ink Objects” beginning on page 5-39, and in the section “Manipulating an Ink Object’s Owner Count” beginning on page 5-41.

GXGetInkTags

You can use the `GXGetInkTags` function to examine one or more of the tag objects associated with an ink object.

```
long GXGetInkTags(gxInk source, long tagType, long index,
                  long count, gxTag items[]);
```

<code>source</code>	A reference to the ink object whose tag list you want to examine.
<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 `GXGetInkTags` function searches the tag list of the `source` ink object for references to tag objects with the tag type specified by the `tagType` parameter. If you specify 0 for the `tagType` parameter, the `GXGetInkTags` function searches all tag types.

You can use the `index` and the `count` parameters to specify which tag references of the appropriate type the `GXGetInkTags` 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.

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

Typically, you call this function once with a `nil` value for the `items` parameter to determine the number of matching tag references. Then you allocate an appropriately sized array and call the function a second time to obtain the references themselves.

Ink Objects

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`ink_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 an ink, use the `GXSetInkTags` function, described in the next section.

GXSetInkTags

You can use the `GXSetInkTags` function to add, remove, or replace tag objects associated with an ink object.

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

<code>target</code>	A reference to the ink object whose tag list you want to alter.
<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 shape's tag list and disposed of.
<code>items</code>	An array of tag references to insert in the tag list.

Ink Objects

DESCRIPTION

The `GXSetInkTags` function allows you add tag references to an ink 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 ink 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.

ERRORS, WARNINGS, AND NOTICES**Errors**

<code>out_of_memory</code>	
<code>ink_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

`index_out_of_range`
`count_out_of_range`

Notices (debugging version)

`tag_already_set`

SEE ALSO

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

To examine the set of tag references associated with an ink, use the `GXGetInkTags` function, described in the previous section.

Getting and Setting an Ink's Color

The functions described in this section allow you to get and set the color structure from a specified ink object, or from the ink object attached to a specified shape.

GXGetInkColor

You can use the `GXGetInkColor` function to examine the color of an ink object.

```
gxColor *GXGetInkColor(gxInk source, gxColor *data);
```

`source` A reference to the ink whose color you want.

`data` A pointer to a color structure. On return, the structure contains the color of the ink object.

function result The color of the source ink object.

DESCRIPTION

The `GXGetInkColor` function returns, as its function result and in the structure pointed to by the `data` parameter, the color of the ink referenced in the `source` parameter.

If the ink object reference or the pointer to the color structure is `nil`, an error is posted, and `nil` is returned as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`ink_is_nil`
`color_is_nil`

SEE ALSO

Ink colors are introduced in the section “Color” beginning on page 5-7, and described fully in the chapter “Colors and Color-Related Objects” in this book.

To assign a color to an ink object, use the `GXSetInkColor` function, described next.

To examine the color of the ink object associated with a shape, use the `GXGetShapeColor` function, described on page 5-70.

GXSetInkColor

You can use the `GXSetInkColor` function to assign a color to an ink object.

```
void GXSetInkColor(gxInk target, const gxColor *data);
```

`target` A reference to the ink to assign the color to.

`data` A pointer to a color structure containing the color to assign to the ink.

DESCRIPTION

The `GXSetInkColor` function assigns the color pointed to by the `data` parameter to the ink object referenced in the `target` parameter. If the color references a color set or color profile object, QuickDraw GX increases the owner count of the referenced object.

SPECIAL CONSIDERATIONS

If the color space of the color pointed to by the `data` parameter is `gxNoSpace`, this function posts a `colorSpace_out_of_range` error.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`ink_is_nil`

`color_is_nil`

`colorSpace_out_of_range`

(debugging version)

`colorSet_access_restricted`

(debugging version)

`colorProfile_access_restricted`

(debugging version)

Notices (debugging version)

`color_already_set`

SEE ALSO

Ink colors are introduced in the section “Color” beginning on page 5-7, and described fully in the chapter “Colors and Color-Related Objects” in this book.

To examine the color of an ink object, use the `GXGetInkColor` function, described in the previous section.

To assign a color to the ink object associated with a shape, use the `GXSetShapeColor` function, described on page 5-71.

GXGetShapeColor

You can use the `GXGetShapeColor` function to examine the color of an ink object associated with a shape.

```
gxColor *GXGetShapeColor(gxShape source, gxColor *data);
```

`source` A reference to the shape whose ink you want the color of.

`data` A pointer to a color structure. On return, the structure contains the color of the ink object associated with the shape.

function result The color of the ink object associated with the source shape object.

DESCRIPTION

The `GXGetShapeColor` function returns, as its function result and in the structure pointed to by the `data` parameter, the color of the ink object associated with the shape object referenced in the `source` parameter.

This call is equivalent to

```
myColor = GXGetInkColor(GXGetShapeInk(myShape), myColor);
```

If the shape object reference or the pointer to the color structure is `nil`, an error is posted, and `nil` is returned as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`color_is_nil`

SEE ALSO

To assign a color to the ink object associated with a shape, use the `GXSetShapeColor` function, described next.

To examine the color of an ink object directly, use the `GXGetInkColor` function, described on page 5-68.

Ink colors are introduced in the section “Color” beginning on page 5-7, and described fully in the chapter “Colors and Color-Related Objects” in this book.

The `GXGetShapeInk` function is described in the chapter “Shape Objects” in this book.

GXSetShapeColor

You can use the `GXSetShapeColor` function to assign a color to the ink object associated with a shape.

```
void GXSetShapeColor(gxShape target, const gxColor *data);
```

target A reference to the shape whose ink object you want to assign the color to.
data A pointer to a color structure containing the color to assign to the shape's ink object.

DESCRIPTION

The `GXSetShapeColor` function assigns the color pointed to by the `data` parameter to the ink object associated with the shape referenced in the `target` parameter.

Calling this function is almost equivalent to

```
GXSetInkColor(GXGetShapeInk(myShape), theColor);
```

except that, if the source shape's ink object is shared with other shapes, `GXSetShapeColor` creates a new copy of that ink object and attaches it to the source shape before changing its color. That way, calling this function does not produce any side effects on other shapes.

If the color pointed to by the `data` parameter references a color set or color profile object, QuickDraw GX increases the owner count of the referenced object.

SPECIAL CONSIDERATIONS

If you use this function to try to assign a color to a bitmap shape, the function posts an `illegal_type_for_shape` error. If the color space of the color pointed to by the `data` parameter is `gxNoSpace`, the function posts a `colorSpace_out_of_range` error.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>color_is_nil</code>	
<code>colorSpace_out_of_range</code>	(debugging version)
<code>colorSet_access_restricted</code>	(debugging version)
<code>colorProfile_access_restricted</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)

Notices (debugging version)

`color_already_set`
`color_is_nil`

Ink Objects

SEE ALSO

To examine the color of the ink object associated with a shape, use the `GXGetShapeColor` function, described in the previous section.

To assign a color to an ink object directly, use the `GXSetInkColor` function, described on page 5-69.

Ink colors are introduced in the section “Color” beginning on page 5-7, and described fully in the chapter “Colors and Color-Related Objects” in this book. Assigning colors to bitmaps is discussed in the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

The `GXGetShapeInk` function is described in the chapter “Shape Objects” in this book.

Getting and Setting an Ink’s Transfer Mode

The functions described in this section allow you to get and set the transfer mode structure from a specified ink object, or from the ink object attached to a specified shape.

GXGetInkTransfer

You can use the `GXGetInkTransfer` function to examine the transfer mode of an ink object.

```
gxTransferMode *GXGetInkTransfer(gxInk source, gxTransferMode
*data);
```

source A reference to the ink whose transfer mode you want.

data A pointer to a transfer mode structure. On return, the structure contains the transfer mode of the ink object.

function result The transfer mode of the source ink object.

DESCRIPTION

The `GXGetInkTransfer` function returns, as its function result and in the structure pointed to by the `data` parameter, the transfer mode of the ink referenced in the `source` parameter.

If the ink object reference or the pointer to the transfer mode structure is `nil`, an error is posted, and `nil` is returned as the function result.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`ink_is_nil`
`transferMode_is_nil`

SEE ALSO

Transfer modes are described in the sections “About Transfer Modes” beginning on page 5-11, and “Working With Transfer Modes” beginning on page 5-44.

To assign a transfer mode to an ink object, use the `GXSetInkTransfer` function, described next.

To examine the transfer mode of the ink object associated with a shape, use the `GXGetShapeTransfer` function, described on page 5-74.

GXSetInkTransfer

You can use the `GXSetInkTransfer` function to assign a transfer mode to an ink object.

```
void GXSetInkTransfer(gxInk target, const gxTransferMode *data);
```

`target` A reference to the ink to assign the transfer mode to.

`data` A pointer to a transfer mode structure containing the transfer mode to assign to the ink.

DESCRIPTION

The `GXSetInkTransfer` function assigns the transfer mode pointed to by the `data` parameter to the ink object referenced in the `target` parameter.

SPECIAL CONSIDERATIONS

The color space of the transfer mode pointed to by the `data` parameter cannot be `gxNoSpace` or any of the packed color spaces (such as, for example, `gxRGB16Space`). If you specify `gxHighlightMode` in some but not all components of the transfer mode, this function posts an `inconsistent_parameters` error.

Ink Objects

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>ink_is_nil</code>	
<code>transferMode_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>invalid_transferMode_colorSpace</code>	(debugging version)
<code>colorSpace_out_of_range</code>	(debugging version)
<code>colorSet_access_restricted</code>	(debugging version)
<code>colorProfile_access_restricted</code>	(debugging version)

SEE ALSO

Transfer modes are described in the sections “About Transfer Modes” beginning on page 5-11, and “Working With Transfer Modes” beginning on page 5-44.

To examine the transfer mode of an ink object, use the `GXGetInkTransfer` function, described in the previous section.

To assign a transfer mode to the ink object associated with a shape, use the `GXSetShapeTransfer` function, described on page 5-75.

Color spaces are described in the chapter “Colors and Color-Related Objects” in this book.

GXGetShapeTransfer

You can use the `GXGetShapeTransfer` function to examine the transfer mode of the ink object associated with a shape.

```
gxTransferMode *GXGetShapeTransfer(gxShape source,
                                   gxTransferMode *data);
```

`source` A reference to the shape whose ink object you want the transfer mode of.

`data` A pointer to a transfer mode structure. On return, the structure contains the transfer mode of the shape’s ink object.

function result The transfer mode of the ink object associated with the source shape object.

Ink Objects

DESCRIPTION

The `GXGetShapeTransfer` function returns, as its function result and in the structure pointed to by the `data` parameter, the transfer mode of the ink object associated with the shape referenced in the `source` parameter.

If the shape object reference or the pointer to the transfer mode structure is `nil`, an error is posted, and `nil` is returned as the function result.

This function is equivalent to

```
theMode = GXGetInkTransfer(GXGetShapeInk(myShape), theMode);
```

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`shape_is_nil`
`transferMode_is_nil`

SEE ALSO

Transfer modes are described in the sections “About Transfer Modes” beginning on page 5-11, and “Working With Transfer Modes” beginning on page 5-44.

To assign a transfer mode to the ink object associated with a shape, use the `GXSetShapeTransfer` function, described next.

To examine the transfer mode of an ink object directly, use the `GXGetInkTransfer` function, described on page 5-72.

The `GXGetShapeInk` function is described in the chapter “Shape Objects” in this book.

GXSetShapeTransfer

You can use the `GXSetShapeTransfer` function to assign a transfer mode to the ink object associated with a shape.

```
void GXSetShapeTransfer(gxShape target, const gxTransferMode
*data);
```

<code>target</code>	A reference to the shape whose ink object you want to assign the transfer mode to.
<code>data</code>	A pointer to a transfer mode structure containing the transfer mode to assign to the shape’s ink.

Ink Objects

DESCRIPTION

The `GXSetShapeTransfer` function assigns the transfer mode pointed to by the `data` parameter to the ink object associated with the shape referenced in the `target` parameter.

Calling this function is almost equivalent to:

```
GXSetInkTransfer ( GXGetShapeInk ( myShape ) , theMode ) ;
```

except that, if the source shape's ink object is shared with other objects, `GXSetShapeTransfer` creates a new copy of that ink object and assigns it to the shape before changing its transfer mode. That way, calling this function does not produce any side effects on other shapes.

SPECIAL CONSIDERATIONS

The color space of the transfer mode pointed to by the `data` parameter cannot be `gxNoSpace` or any of the packed color spaces (such as, for example, `gxRGB16Space`). If you specify `gxHighlightMode` in some but not all components of the transfer mode, this function posts an `inconsistent_parameters` error.

ERRORS, WARNINGS, AND NOTICES**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>transferMode_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>invalid_transferMode_colorSpace</code>	(debugging version)
<code>colorSet_access_restricted</code>	(debugging version)
<code>colorSpace_out_of_range</code>	(debugging version)
<code>colorProfile_access_restricted</code>	(debugging version)

SEE ALSO

Transfer modes are described in the sections “About Transfer Modes” beginning on page 5-11, and “Working With Transfer Modes” beginning on page 5-44.

To examine the transfer mode of the ink object associated with a shape, use the `GXGetShapeTransfer` function, described in the previous section.

To assign a transfer mode to an ink object directly, use the `GXSetInkTransfer` function, described on page 5-73.

The `GXGetShapeInk` function is described in the chapter “Shape Objects” in this book.

Summary of Ink Objects

Constants and Data Types

The Ink Object

```
typedef struct gxPrivateInkRecord *gxInk;
```

Ink Attributes

```
enum gxInkAttributes{
    gxPortAlignDitherInk = 0x0001,
    gxForceDitherInk      = 0x0002,
    gxSuppressDitherInk   = 0x0004,
    gxSuppressHalftoneInk= 0x0008
};
```

```
typedef long gxInkAttribute;
```

The Color Structure

```
struct gxColor{
    gxColorSpace space;
    gxColorProfile profile;
    union {
        gxCMYKColor      cmyk;
        gxRGBColor        rgb;
        gxRGBAColor       rgba;
        gxHSVColor        hsv;
        gxHLSColor        hls;
        gxCIEColor         cie;
        gxYIQColor         yiq;
        gxColorValue       gray;
        gxGrayAColor       graya;
        unsigned short     pixel16;
        unsigned long      pixel32;
        gxIndexedColor     indexed;
        gxColorValue       component[4];
    } element;
};
```

```
typedef unsigned char gxComponentMode;
```

Ink Objects

The Transfer Mode Structure

```

struct gxTransferMode{
    gxColorSpace          space;
    gxColorSet            set;
    gxColorProfile        profile;
    Fixed                 sourceMatrix[5][4];
    Fixed                 deviceMatrix[5][4];
    Fixed                 resultMatrix[5][4];
    gxTransferFlag        flags;
    struct gxTransferComponent component[4];
};

```

Transfer Mode Flags

```

enum gxTransferFlags{
    gxRejectSourceTransfer = 0x0001,    /* At least one source component
                                         must be out of range */
    gxRejectDeviceTransfer = 0x0002,    /* At least one device component
                                         must be out of range */
    gxSingleComponentTransfer= 0x0004    /* duplicate gxTransferComponent[0]
                                         for all components in transfer */
};

typedef long gxTransferFlag;

```

The Transfer Component Structure

```

struct gxTransferComponent{
    gxComponentMode    mode;
    gxComponentFlag    flags;
    gxColorValue        sourceMinimum;
    gxColorValue        sourceMaximum;
    gxColorValue        deviceMinimum;
    gxColorValue        deviceMaximum;
    gxColorValue        clampMinimum;
    gxColorValue        clampMaximum;
    gxColorValue        operand;
};

```

Ink Objects

Component Modes

```
enum gxComponentModes{
    gxNoMode = 0,
    gxCopyMode,
    gxAddMode,
    gxBlendMode,
    gxMigrateMode,
    gxMinimumMode,
    gxMaximumMode,
    gxHighlightMode,
    gxAndMode,
    gxOrMode,
    gxXorMode,
    gxRampAndMode,
    gxRampOrMode,
    gxRampXorMode,
    gxOverMode,
    gxAtopMode,
    gxExcludeMode,
    gxFadeMode
};
```

Transfer Component Flags

```
enum gxComponentFlags{
    gxOverResultComponent= 0x01, /* AND the result component with
                                   0xFFFF before clamping */
    gxReverseComponent    = 0x02 /* Reverse source and device components
                                   before applying transfer mode */
};

typedef unsigned char gxComponentFlag;
```

Functions

Creating and Manipulating Ink Objects

```
gxInk GXNewInk                (void);
void GXDisposeInk              (gxInk target);
gxInk GXCopyToInk              (gxInk target, gxInk source);
boolean GXEqualInk              (gxInk one, gxInk two);
gxInk GXCloneInk               (gxInk source);
```

Ink Objects

Manipulating Ink Object Properties

```

void GXResetInk                (gxInk target);
gxInkAttribute GXGetInkAttributes
                                (gxInk source);
void GXSetInkAttributes        (gxInk target, gxInkAttribute attributes);
gxInkAttribute GXGetShapeInkAttributes
                                (gxShape source);
void GXSetShapeInkAttributes (gxShape target, gxInkAttribute attributes);
long GXGetInkOwners            (gxInk source);
long GXGetInkTags              (gxInk source, long tagType, long index,
                                long count, gxTag items[]);
void GXSetInkTags              (gxInk target, long tagType, long index,
                                long oldCount, long newCount,
                                const gxTag items[]);

```

Getting and Setting an Ink's Color

```

gxColor *GXGetInkColor         (gxInk source, gxColor *data);
void GXSetInkColor             (gxInk target, const gxColor *data);
gxColor *GXGetShapeColor       (gxShape source, gxColor *data);
void GXSetShapeColor           (gxShape target, const gxColor *data);

```

Getting and Setting an Ink's Transfer Mode

```

gxTransferMode *GXGetInkTransfer
                                (gxInk source, gxTransferMode *data);
void GXSetInkTransfer          (gxInk target, const gxTransferMode *data);
gxTransferMode *GXGetShapeTransfer
                                (gxShape source, gxTransferMode *data);
void GXSetShapeTransfer        (gxShape target, const gxTransferMode *data);

```