

Style Objects

This chapter describes style objects and the functions you can use to manipulate them. Read this chapter if you create or use any kind of style objects for the QuickDraw GX shapes you create.

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

For more information on style objects for graphic shapes, see the geometric styles chapter of *Inside Macintosh: QuickDraw GX Graphics*. For more information on style objects for typographic shapes, see the typographic styles chapter of *Inside Macintosh: QuickDraw GX Typography*.

This chapter introduces QuickDraw GX style objects and describes their properties. It then shows how to use general QuickDraw GX style-manipulation functions to

- create and manipulate style objects
- manipulate style object properties

This chapter also lists and cross-references all style-related QuickDraw GX functions that are described elsewhere in this book and in other parts of *Inside Macintosh*.

About Style Objects

A style object exists to provide information about a shape. Each QuickDraw GX shape consists of a shape object, a style object, an ink object, and a transform object; the style object associated with a shape defines much of the shape’s appearance, such as the size of the pen with which it is drawn or the size of its text.

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

Styles are device independent. Their information is not affected by the properties of the display device to which the shapes they modify are drawn.

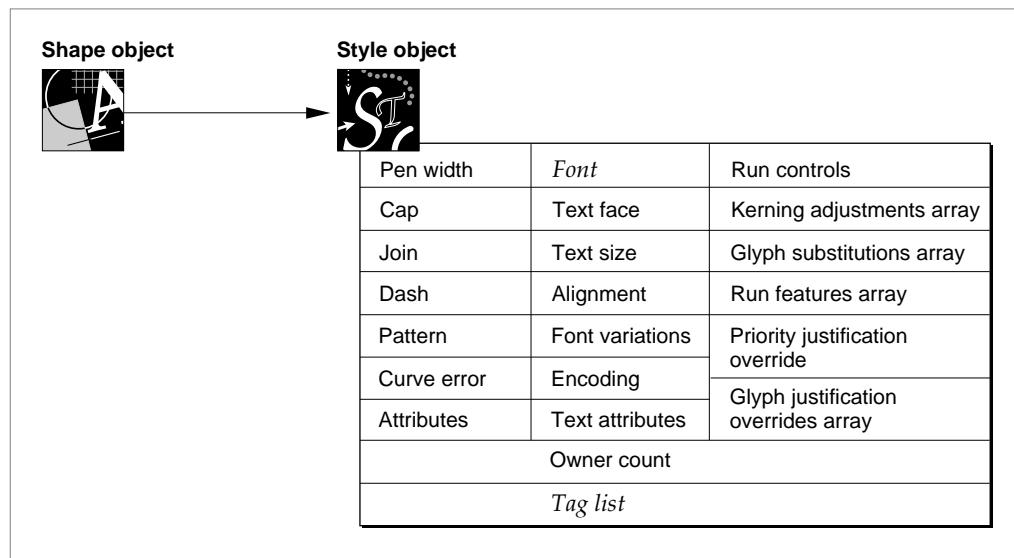
There are three categories of information that style objects contain: graphic, typographic, and common. The graphic information applies to style objects associated with graphic shapes, the typographic information applies to style objects associated with typographic shapes, and the common information applies to both. Because the information is stored separately, the same style object can apply to both kinds of shapes. The QuickDraw GX object architecture allows you to perform several operations on a style object without regard for what kind it is; those are the operations described in this chapter. Features and operations specific to styles for graphic shapes are described in *Inside Macintosh: QuickDraw GX Graphics*; those specific to styles for typographic shapes are described in *Inside Macintosh: QuickDraw GX Typography*.

Style Object Properties

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

Style objects have 22 accessible properties, as shown in Figure 3-1. The properties are grouped into columns that reflect the category of shape that uses them. Note that, because a style is an object and not a data structure, the order of the properties as shown in Figure 3-1 is completely arbitrary. Properties in italics are references to other objects.

Figure 3-1 The style object and its properties



Seven properties pertain mostly to style objects associated with graphic shapes:

- ***Pen width***. The width of the pen used to draw the shape.
- ***Cap***. The shape (such as an arrowhead, or any other geometric shape) to draw at the start and end of each contour in the shape.
- ***Join***. The appearance (such as rounded or sharp, or any other geometric shape) of corners where a shape's lines or contours meet.
- ***Dash***. The appearance of dashed lines or contours in a shape. The dashing capability is very general in QuickDraw GX; you can specify any geometric shape, or even a sequence of glyphs, for a dash.
- ***Pattern***. The pattern (actually, any geometric shape, glyph shape, or bitmap shape) to use in filling the geometry of the shape.
- ***Curve error***. The allowable error for operations such as converting a path shape to a polygon shape.

Style Objects

- **Attributes.** A set of flags that allow you to specify how QuickDraw GX places the pen and whether the shape is constrained to a grid when drawn. (The grid-constraining attributes can apply to typographic shapes also.)

Thirteen of the style object's properties pertain only to styles associated with typographic shapes. The portion of a typographic shape to which a style object applies is called a **style run**. The first seven typographic style properties apply, for the most part, to all typographic shapes:

- **Font.** The reference to the font to use in drawing the text of this style run. (In QuickDraw GX, a font is an object.)
- **Text face.** The text face—the constructed stylistic variation from plain text—to apply when drawing the text of this style run.
- **Text size.** The size, in typographic points (72 per inch), to draw the text of this style run.
- **Alignment.** The alignment value to use when drawing the text of this style run. Text may be left-aligned, right-aligned, anywhere between the two alignments (such as centered), or fully justified. (This property is not used by layout shapes).
- **Font variations.** The list of font variations—stylistic variations built into the font—specified for drawing the text of this style run.
- **Encoding.** The type of character encoding used to represent the text of this style run, as well as its script and language.
- **Text attributes.** A set of flags that allow you to specify how QuickDraw GX alters glyph outlines or chooses the proper metrics for horizontal or vertical text.

The remaining six of the thirteen typographic style properties apply to layout shapes only:

- **Run controls.** A set of values and flags that control various aspects of how the text in this style run is displayed.
- **Kerning adjustments array.** An array specifying changes to the font-specified kerning (positional adjustment) for pairs of glyphs in this style run.
- **Glyph substitutions array.** An array specifying substitute glyphs for those that would normally be displayed in this style run.
- **Run features array.** An array specifying the set of font features—typographic capabilities as defined by the font—to apply to the text of this style run.
- **Priority justification override.** A structure that redefines the justification priorities and behaviors for whole classes of glyphs.
- **Glyph justification overrides array.** An array that redefines the justification priorities and behaviors for individual glyphs.

Style Objects

The two remaining style object properties pertain to all styles, for all shapes:

- **Owner count.** The number of existing references to this style object.
- **Tag list.** A list of references to custom information about this style 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 style object properties. Table 3-1 shows where to go for that information, depending on the type of shape object that uses the style.

Table 3-1 Where to go for information on style object properties and functions

| For style objects used by... | Look in... |
|------------------------------|---|
| Graphic shapes | Geometric styles chapter of <i>QuickDraw GX Graphics</i> |
| All typographic shapes | Typographic styles chapter of <i>QuickDraw GX Typography</i> (For style attributes that can apply to typographic shapes, see also the geometric styles chapter of <i>QuickDraw GX Graphics</i>) |
| Layout shapes only | Layout styles and layout line control chapters of <i>QuickDraw GX Typography</i> |
| All shapes | This chapter |

As Table 3-1 shows, most style-object properties and functions are described elsewhere. Only those properties that pertain to all shapes—the owner count and tag list, and the functions that manipulate them—are described in this chapter.

The Default Style Object

When QuickDraw GX first creates a style object, that object has default characteristics defined by QuickDraw GX. Every default style object has the following properties:

- an empty tag list
- an owner count of 1

All other properties are zero or `nil`, except that the value of the text size property is 12.0, and the scale value within the dash property is 1.0. The font property is `nil`, which means that QuickDraw GX uses the default font in drawing text; however, your application can control what font is used for the default. See the font objects chapter in *Inside Macintosh: QuickDraw GX Typography* for more information.

Style Objects

Unlike shape objects, whose default properties vary with type, there is only one single default style object for QuickDraw GX. If the shape objects you create reference the default style object, you need to explicitly set all graphic or typographic properties for that style after you create the shape. Also unlike shape objects, you cannot change the definition of the default style object. However, you can create a style object with specific properties, and then change the definition of the default shape object so that newly created shapes reference that customized style object.

Using Style Objects

This section describes the basic style-creation and style-manipulation capabilities that QuickDraw GX provides, capabilities that are independent of the specific type of style object involved. For detailed information on using styles of specific types, see the appropriate chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*.

This section describes how you can

- create and manipulate style objects
- manipulate certain style object properties

Creating and Manipulating Style Objects

This section describes how you can create and interact with style objects as whole entities—to create, dispose of, copy, compare, and clone them. Manipulating the individual properties of style objects is described under “Manipulating Style Object Properties” beginning on page 3-10.

Creating and Deleting a Style Object

QuickDraw GX provides the `GXNewStyle` function to allow you to create a new style object. Before you can create a style object, you need to be in the QuickDraw GX environment. However, if you are not already in the QuickDraw GX environment, `GXNewStyle` calls the necessary functions for you. The functions for controlling memory use in the QuickDraw GX environment are described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Note that you can also create a new style object by copying an existing one: see the section “Copying, Comparing, and Cloning Style Objects” beginning on page 3-8.

To delete your application’s reference to a style object, call the `GXDisposeStyle` function. Calling `GXDisposeStyle` may or may not actually release the memory allocated for that style object, depending on the style’s owner count. `GXDisposeStyle` decreases the style object’s owner count by 1; if that brings the owner count to zero, the style is completely deleted and its memory released. See “Manipulating a Style Object’s Owner Count” beginning on page 3-11.

Style Objects

Owner counts and what it means to dispose of an object are described in general in the chapter “Introduction to Objects” in this book.

The following code fragment defines and creates the style object `myStyle`, sets some of its properties, and disposes of it:

```
gxStyle      myStyle;
.
.
.
myStyle = GXNewStyle ();
GXSetStylePen(myStyle, ff(2));
GXSetStyleAttributes(myStyle, gxOutsideFrameStyle);
.
.
.
if (myStyle != nil) GXDisposeStyle(myStyle);
```

The `GXNewStyle` function is described on page 3-17. The `GXDisposeStyle` function is described on page 3-17.

Copying, Comparing, and Cloning Style Objects

You can use the `GXCopyToStyle` function to copy information from one style object to another or to create a new copy of a style object.

Listing 3-1 is a code fragment that changes the type of the shape `myShape` to a glyph shape with four distinct style runs, and then fills out the style list, an array of style-object references in the geometry of the glyph shape. (Glyph shapes and layout shapes have style-object references in their geometries in addition to the style property that every shape object has.) The code creates new copies of the style object originally referenced by `myShape`, each time assigning the style reference to a position in the `styleSet` array and then modifying some of the style’s properties. Finally, the code assigns the style list to the shape geometry.

The code in Listing 3-1 uses the library functions `SetStyleCommonFont` and `SetStyleCommonFace` to modify the font and text-face properties of the style objects it creates by copying. The text is defined in the string `str`, and the lengths of the style runs are defined in the `runs` array. (Each style run is defined to be one glyph long in this sample.)

Listing 3-1 Building a style list by copying a style object

```

GXSetShapeType(myShape, gxGlyphType);

/* use the default shape's style for first style run */
styleSet[0] = nil;

/* use condensed Helvetica for the second style run */
styleSet[1] = GXCopyToStyle(nil, GXGetShapeStyle(myShape));
SetStyleCommonFont(styleSet[1], helveticaFont);
SetStyleCommonFace(styleSet[1], gxCondense);

/* use extended Times for the third style run */
styleSet[2] = GXCopyToStyle(nil, GXGetShapeStyle(myShape));
SetStyleCommonFont(styleSet[2], timesFont);
SetStyleCommonFace(styleSet[2], gxExtend);

/* use 20-pt. italic Helvetica for the fourth style run */
styleSet[3] = GXCopyToStyle(nil, GXGetShapeStyle(myShape));
SetStyleCommonFont(styleSet[3], helveticaFont);
SetStyleCommonFace(styleSet[3], gxItalic);
GXSetStyleTextSize(styleSet[3], ff(20));

/* set the size (number of glyphs) of each style run */
for (counter = 0; counter < strlen(str); counter++) {
    runs[counter] = 1;                /* each run is 1 glyph long */
    styles[counter] = styleSet[counter & 3];
}

/* assign the styles array to the style list */
GXSetGlyphs(myShape, nil, nil, nil, nil, nil, runs, styles);

```

You can test if two style-object references refer to the same style object by simply testing the references for equality. You can also compare two different style objects for equality with the `GXEqualStyle` function. For two style objects to be equal, their graphic and typographic properties must have identical values, although their general object properties (owner count and tag list) do not need to be identical. Note that style object copies created with the `GXCopyToStyle` function are always equal to the style from which they were copied.

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

Style Objects

Functionally, `GXCloneStyle` does nothing more than increase the owner count of a style object. You can clone a style with a statement such as the following:

```
aStyleClone = GXCloneStyle(aStyle);
```

This code has almost the same effect as

```
aStyleClone = aStyle;
```

that is, it sets the `aStyleClone` variable to reference the same style object as the `aStyle` variable. The difference is that `GXCloneStyle` also increments the style's owner count.

For more information about cloning objects, see the chapter “Introduction to QuickDraw GX” in this book. For information on manipulating style owner counts, including examples of cloning styles, see the section “Manipulating a Style Object's Owner Count” beginning on page 3-11 of this chapter.

The `GXCopyToStyle` function is described on page 3-18. The `GXEqualStyle` function is described on page 3-19. The `GXCloneStyle` function is described on page 3-20.

Loading and Unloading Style Objects

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

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

Manipulating Style Object Properties

Once you have created a style object, you can customize some of its features using the techniques described in this section. However, most of the functions you use to set style properties are described in the chapters that discuss style objects in *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*.

This section describes how to manipulate those properties of style objects that are independent of the type of shape the style is associated with. You can reset a style's properties back to their default values, you can determine the owner count, and you can get and set the tag list.

For manipulating style objects as a whole, see “Creating and Manipulating Style Objects” beginning on page 3-7.

Resetting a Style Object's Default Properties

When you create a new style object with the `GXNewStyle` function, QuickDraw GX creates a style object with default properties. If you have altered any of the style object's properties using functions described in this chapter or in *Inside Macintosh: QuickDraw GX Graphics* or *Inside Macintosh: QuickDraw GX Typography*, you can reset the properties back to their default values using the `GXResetStyle` function.

Calling `GXResetStyle` returns all of the style's graphic and typographic properties to their default values. It does not affect the style's owner count or tag list.

The `GXResetStyle` function is described on page 3-21.

Getting and Setting Style Attributes and Text Attributes

A style object has two separate properties, *attributes* and *text attributes*, that consist of flags that affect style behavior. The attributes property of a style object affects mostly graphic shapes. You retrieve and assign attribute values, such as pen width, with the `GXGetStyleAttributes` and `GXSetStyleAttributes` functions. These functions, and the style attributes themselves, are described in the geometric styles chapter of *Inside Macintosh: QuickDraw GX Graphics*.

The text attributes property of a style object affects typographic shapes only. You retrieve and assign text attribute values, such as vertical-text selection, with the `GXGetStyleTextAttributes` and `GXSetStyleTextAttributes` functions, respectively. These functions, and the text attributes themselves, are described in the typographic styles chapter of *Inside Macintosh: QuickDraw GX Typography*.

Manipulating a Style 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 style object, QuickDraw GX sets the owner count of the new style to 1. When you assign an existing style object to a shape, QuickDraw GX increments the style's owner count to correspond to the new reference to the style contained in the shape object.

If you want to manage a style's owner count directly—for example, if you want to track object references that you place in your own data structures, or if you want to know whether a style object is shared—you can use the `GXGetStyleOwners` function to determine the owner count of a style, and the `GXCloneStyle` and `GXDisposeStyle` functions to change the owner count of a style. The `GXCloneStyle` function increments the style's owner count, and the `GXDisposeStyle` function decrements the style's owner count, freeing the memory used by the style if the owner count goes to 0.

The `GXGetStyleOwners` function is described on page 3-22.

The following subsections discuss 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 other shared objects.

Style Objects

Avoiding Excessive Owner Count

The following is one plausible, but incorrect, way to create a style object and assign it to (the style reference property of) a shape:

```
GXSetShapeStyle(myShape, GXNewStyle());
```

After the execution of this statement, the owner count of the just-created style object is 2, not 1; creating the style object initialized its owner count to 1, and assigning it to the shape incremented its owner count to 2. If you were unaware of that, and deleted the shape object with the statement

```
GXDisposeShape(myShape);
```

the owner count of the style object would be decremented to 1, and the style would be left allocated in the heap when it should have been deleted.

A better way to create and assign a style object is to allocate a variable and use it in the assignment:

```
myStyle = GXNewStyle();
```

```
GXSetShapeStyle(myShape, myStyle);
```

As before, the style object's owner count is now 2. When you are finished with the variable reference to the style object, you can dispose of it:

```
GXDisposeStyle(myStyle);
```

That decreases the style's owner count to 1. When you are finished with the shape object, dispose of it as before:

```
GXDisposeShape(myShape);
```

That decreases the style's owner count to 0, and the style object is deleted as intended.

Avoiding Insufficient Owner Count

The following is one plausible, but incorrect, way to temporarily assign a style object to a shape, referenced in this example by the variable `myShape`. These statements save the original style into a variable, create a new style object, and assign the new style to the shape:

```
gxStyle myOldStyle = GXGetShapeStyle(myShape);
gxStyle myNewStyle = GXNewStyle();
GXSetShapeStyle(myShape, myNewStyle);
```

Style Objects

The first statement does not increase the owner count of the style referenced by `myOldStyle`; no new object is created and no additional references to `myShape` exist in any object. The second statement results in an owner count of 1 for the style referenced by `myNewStyle`. The third statement decrements the owner count of the style referenced by `myOldStyle`, and increments the owner count of the style referenced by `myNewStyle` (from 1 to 2).

Now suppose that you manipulate the new style object, draw the shape, and then wish to dispose of the new style and reassign the original style object back to the shape. You might expect to make two statements like this:

```
GXDisposeStyle(myNewStyle);
GXSetShapeStyle(myShape, myOldStyle);
```

As you would expect, disposing of `myNewStyle` decrements the owner count of the new style object from 2 to 1, and calling `GXSetShapeStyle` further decrements the owner count of the new style from 1 to 0, so that QuickDraw GX can delete it. However, the original style object, referenced by `myOldStyle`, may have been deleted by the original call to `GXSetShapeStyle` (because its owner count may have gone to 0 as a result of the call). If it has, `myOldStyle` will be `nil` and the new call to `GXSetShapeStyle` will fail.

A better way to temporarily save and restore a style object is to clone the original style before assigning the new style, as follows:

```
gxStyle myOldStyle = GXGetShapeStyle(myShape);
gxStyle myNewStyle = GXNewStyle();
GXCloneStyle(myOldStyle);
GXSetShapeStyle(myShape, myNewStyle);
```

The result of these statements is (assuming no other references to the style objects) an owner count of 2 for both the original and new style objects. Then, when the time comes to restore the original style object to the shape, you can make these statements:

```
GXDisposeStyle(myNewStyle);
GXSetShapeStyle(myShape, myOldStyle);
GXDisposeStyle(myOldStyle);
```

The first statement decrements the owner count of the new style from 2 to 1; the second statement decrements it from 1 to 0. The second statement increments the owner count of the original style from 1 to 2, so the third statement is added to bring it back down to 1, its original value.

Style Objects

Getting and Setting a Style Object's Tag References

You can examine the list of references to tag objects currently associated with a style object using the `GXGetStyleTags` function. Once you create a tag object, you can attach it to a style object using the `GXSetStyleTags` function. You can attach as many tag objects as you like to a style 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 `GXGetStyleTags` function is described on page 3-22. The `GXSetStyleTags` function is described on page 3-24.

Style-Related Functions Described Elsewhere

Table 3-2 lists functions whose names contain the word `Style` that are either not described in this chapter or are described in more detail elsewhere. For each book and chapter, the table lists the style-related functions described in that chapter.

Table 3-2 Style-related functions described elsewhere

| Book and chapter | Functions described |
|--|--|
| <i>Inside Macintosh: QuickDraw GX Graphics</i> | |
| “Geometric Styles” | <code>GXGetStylePen</code> <code>GXSetStylePen</code> <code>GXGetStyleCap</code> <code>GXSetStyleCap</code> <code>GXGetStyleJoin</code> <code>GXSetStyleJoin</code> <code>GXGetStyleDash</code> <code>GXGetStyleDash</code> <code>GXGetStylePattern</code> <code>GXSetStylePattern</code> <code>GXGetStyleCurveError</code> <code>GXSetStyleCurveError</code> <code>GXGetStyleAttributes</code> <code>GXSetStyleAttributes</code> |

Table 3-2 Style-related functions described elsewhere (continued)

| Book and chapter | Functions described |
|--|---|
| <i>Inside Macintosh: QuickDraw GX Typography</i> | |
| “Typographic Styles” | GXGetStyleFont GXSetStyleFont GXGetStyleFontMetrics GXGetStyleFace GXSetStyleFace GXGetStyleTextSize GXSetStyleTextSize GXGetStyleJustification GXSetStyleJustification GXGetStyleFontVariations GXSetStyleFontVariations GXGetStyleFontVariationSuite GXGetStyleEncoding GXSetStyleEncoding GXGetStyleTextAttributes GXSetStyleTextAttributes |
| “Layout Styles” | GXGetStyleRunControls GXSetStyleRunControls GXGetStyleRunKerningAdjustments GXSetStyleRunKerningAdjustments GXGetStyleRunGlyphSubstitutions GXSetStyleRunGlyphSubstitutions GXGetStyleRunFeatures GXSetStyleRunFeatures |
| “Layout Line Control” | GXGetStyleRunPriorityJustOverride GXSetStyleRunPriorityJustOverride GXGetStyleRunGlyphJustOverrides GXSetStyleRunGlyphJustOverrides |

Style Objects Reference

This section provides reference information about the data structures and functions that allow you to create and manipulate style objects and alter their properties. It includes

- a type definition of the data type that applies to style objects in general
- descriptions of the QuickDraw GX functions that operate on style objects in general, independent of the type of shape involved

Constants and Data Types

This section describes the data type that you use to gain access to style objects.

Style-related QuickDraw GX constants and data types not described in this section are related to geometric and typographic shapes, and are thus described in the geometric styles chapter of *Inside Macintosh: QuickDraw GX Graphics* and the typographic styles, layout styles, and layout line control chapters of *Inside Macintosh: QuickDraw GX Typography*.

The Style Object

QuickDraw GX provides you with access to an individual style object through a `gxStyle` reference:

```
typedef struct gxPrivateStyleRecord *gxStyle;
```

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

Functions

This section describes the QuickDraw GX functions you can use to

- create and manipulate a style object
- manipulate the general object properties of a style object

Note

Style-related QuickDraw GX functions not described in this section are described in the chapters listed and cross-referenced in Table 3-2 on page 3-14. ♦

Creating and Manipulating Style Objects

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

To associate a style object with a QuickDraw GX shape object, use the `GXGetShapeStyle` and `GXSetShapeStyle` functions, described in the chapter “Shape Objects” in this book.

GXNewStyle

You can use the `GXNewStyle` function to create a new style object with default properties.

```
gxStyle GXNewStyle(void);
```

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

DESCRIPTION

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

- An empty tag list.
- An owner count of 1.
- A text size of 12.0.
- A scale value within the dash property of 1.0.
- No font specified.

All other properties are zero or `nil`.

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewStyle` function creates a style 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 style values are described in the section “The Default Style Object” on page 3-6.

GXDisposeStyle

You can use the `GXDisposeStyle` function to release a reference to a style object.

```
void GXDisposeStyle(gxStyle target);
```

target A reference to the style object to dispose of.

Style Objects

DESCRIPTION

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

ERRORS, WARNINGS, AND NOTICES**Errors**

`style_is_nil`

SEE ALSO

Owner counts for style objects are discussed in the section “Copying, Comparing, and Cloning Style Objects” on page 3-8, and in the section “Manipulating a Style Object’s Owner Count” beginning on page 3-11. To examine the owner count of a style, use the `GXGetStyleOwners` function, described on page 3-22.

GXCopyToStyle

You can use the `GXCopyToStyle` function to create a copy of an existing style object.

```
gxStyle GXCopyToStyle(gxStyle target, gxStyle source);
```

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

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

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

DESCRIPTION

The `GXCopyToStyle` function copies the contents of an existing style object to another, or it creates a new style object and copies the contents of an existing style object into it. The function copies the stylistic and typographic properties and the tag list (but not the owner count) of the style object specified by the `source` parameter into the style 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 `GXCopyToStyle` function creates a new style object and copies the source properties, including tag list, into it. The function gives the new style object an owner count of 1.

You can use the `GXCopyToStyle` function to create a copy of a style object in order to modify it without changing the original.

SPECIAL CONSIDERATIONS

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

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`style_is_nil`

SEE ALSO

To create a new style that is a copy of the default style instead of a copy of an existing style, use the `GXNewStyle` function, described on page 3-17.

To compare two style objects, use the `GXEqualStyle` function, described in the next section.

GXEqualStyle

You can use the `GXEqualStyle` function to determine if two style objects are equal.

```
boolean GXEqualStyle(gxStyle one, gxStyle two);
```

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

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

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

DESCRIPTION

The `GXEqualStyle` function returns as its function result a Boolean value indicating whether the two style objects are equal.

For two style objects to be equal, they must have identical properties, except that their common object properties (owner count and tag list) need not be identical.

Style Objects

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
style_is_nil

SEE ALSO

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

GXCloneStyle

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

```
gxStyle GXCloneStyle(gxStyle source);
```

`source` A reference to the style to clone.

function result A reference to the cloned style.

DESCRIPTION

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

This function returns as its function result a reference to the style—the same reference you pass in as the `source` parameter. The only other action that `GXCloneStyle` performs is to increment the style’s owner count.

ERRORS, WARNINGS, AND NOTICES

Errors

style_is_nil

SEE ALSO

Owner counts for style objects are discussed in the section “Copying, Comparing, and Cloning Style Objects” beginning on page 3-8, and in the section “Manipulating a Style Object’s Owner Count” beginning on page 3-11.

To examine the owner count of a style, use the `GXGetStyleOwners` function, described on page 3-22. To decrement the owner count of a style, use the `GXDisposeStyle` function, described on page 3-17.

Manipulating Style Object Properties

This section describes the functions that allow you to manipulate certain properties of style objects—those properties that are independent of the kind of style object. The functions in this section allow you to reset some of the properties of a style object to their default values, find a style object's owner count, and manipulate a style object's tag list.

Functions that allow you to manipulate graphics-specific style properties are described in the geometric styles chapter of *Inside Macintosh: QuickDraw GX Graphics*; functions that allow you to manipulate typographic-specific style properties are described in the typographic styles chapter and layout styles chapter of *Inside Macintosh: QuickDraw GX Typography*.

GXResetStyle

You can use the `GXResetStyle` function to reset the properties of an existing style object to their default values.

```
void GXResetStyle(gxStyle target);
```

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

DESCRIPTION

The `GXResetStyle` function resets all properties of the target style object, except owner count and tag list, to their default values. The owner count and tag list are unaffected.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

SEE ALSO

Default properties of style objects are discussed in the section “The Default Style Object” on page 3-6.

GXGetStyleOwners

You can use the `GXGetStyleOwners` function to determine the number of references to a particular style.

```
long GXGetStyleOwners(gxStyle source);
```

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

function result The owner count of the source style.

DESCRIPTION

The `GXGetStyleOwners` function returns as its function result the owner count of the style specified by the `source` parameter. The owner count is the current number of references to the style object.

ERRORS, WARNINGS, AND NOTICES

Errors

`style_is_nil`

SEE ALSO

Owner counts are discussed in the section “Copying, Comparing, and Cloning Style Objects” on page 3-8, and in the section “Manipulating a Style Object’s Owner Count” beginning on page 3-11.

To increment the owner count of a style, use the `GXCloneStyle` function, described on page 3-20. To decrement the owner count of a style, use the `GXDisposeStyle` function, described on page 3-17.

GXGetStyleTags

You can use the `GXGetStyleTags` function to examine one or more of the tag objects associated with a style object.

```
long GXGetStyleTags(gxStyle source, long tagType, long index,  
                    long count, gxTag items[]);
```

`source` A reference to the style object to examine the tag list of.

`tagType` The type of tag object to search for. A value of 0 indicates that you want to look for all tag types.

Style Objects

| | |
|--------------------|--|
| <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 `GXGetStyleTags` function searches the tag list of the source style object for references to tag objects with the tag type specified by the `tagType` parameter. If you specify 0 for the `tagType` parameter, the `GXGetShapeTags` function searches all tag types.

You can use the `index` and the `count` parameters to specify which tag references of the appropriate type the `GXGetStyleTags` 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 `GXGetStyleTags` function returns in it the tag references that were found.

ERRORS, WARNINGS, AND NOTICES**Errors**

| | |
|-------------------------------------|---------------------|
| <code>out_of_memory</code> | |
| <code>style_is_nil</code> | |
| <code>index_is_less_than_one</code> | (debugging version) |
| <code>count_is_less_than_one</code> | (debugging version) |

Warnings

| |
|---------------------------------|
| <code>index_out_of_range</code> |
| <code>count_out_of_range</code> |

SEE ALSO

Tag objects are introduced in the chapter “Introduction to Objects” in this book. Functions to create and manipulate tags objects, and a list of reserved tag types, are described in the chapter “Tag Objects” in this book.

To change the set of tag references associated with a style, use the `GXSetStyleTags` function, described in the next section.

GXSetStyleTags

You can use the `GXSetStyleTags` function to add, remove, or replace tag objects associated with a style object.

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

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

DESCRIPTION

The `GXSetStyleTags` function allows you add tag references to a style 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 style object to be modified, the `newCount` parameter specifies the number of tag references to add, and the `items` parameter provides the new tag reference.

- 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.)

Style Objects

- 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>style_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 introduced in the chapter “Introduction to Objects” in this book. Functions to create and manipulate tags objects, and a list of reserved tag types, are described in the chapter “Tag Objects” in this book.

To examine the set of tag references associated with a style, use the `GXGetStyleTags` function, described in the previous section.

Summary of Style Objects

Constants and Data Types

The style object

```
typedef struct gxPrivateStyleRecord *gxStyle;
```

Functions

Creating and Manipulating Style Objects

```
gxStyle GXNewStyle          (void);
void GXDisposeStyle        (gxStyle target);
gxStyle GXCopyToStyle      (gxStyle target, gxStyle source);
boolean GXEqualStyle       (gxStyle one, gxStyle two);
gxStyle GXCloneStyle       (gxStyle source);
```

Manipulating Style Object Properties

```
void GXResetStyle          (gxStyle target);
long GXGetStyleOwners      (gxStyle source);
long GXGetStyleTags        (gxStyle source, long tagType, long index,
                             long count, gxTag items[]);
void GXSetStyleTags        (gxStyle target, long tagType, long index,
                             long oldCount, long newCount,
                             const gxTag items[]);
```