

Geometric Styles

This chapter describes the geometric properties of style objects, which you can use to apply certain types of stylistic variations to QuickDraw GX shapes. In particular, this chapter shows how you can

- constrain the drawing of a shape to a grid
- specify the pen width to use when drawing a shape's frame
- indicate the placement of the pen relative to the shape's frame
- specify what to draw at the beginnings and the ends of a shape's contours
- specify what to draw at the corners of a shape's contours
- dash the contours of a shape
- fill a shape, or the frame of a shape, with a pattern

You can also apply stylistic variations to typographic shapes, using the typographic properties of the style object. For example, you can use the style associated with a text shape to specify the text's font, font size, and type style. The chapter "Typographic Styles" in *Inside Macintosh: QuickDraw GX Typography* discusses the text-related properties of style objects.

You should be familiar with some of the information in *Inside Macintosh: QuickDraw GX Objects* before you read this chapter; in particular, you should read the chapters "Introduction to QuickDraw GX Objects" and "Style Objects" in that book.

About Geometric Styles

A style is a group of stylistic variations applied to a shape. QuickDraw GX provides two major categories of stylistic variations: geometric variations, which include pen width, dashes, patterns, and so on, and typographic variations, which include font, font size, typestyle, and so on.

Both types of stylistic variation are encapsulated in a **style object**. Like a shape object, a style object is a data structure that you manipulate with functions provided by QuickDraw GX. Each style object has a group of properties, and each **style property** represents a different stylistic variation.

Shapes and Styles


In general, a shape object is an object with a group of properties that describe a geometry; a style object is an object with a group of properties that affect how QuickDraw GX interprets a shape's geometry during drawing.

Geometric Styles

Every QuickDraw GX shape object contains a reference to a style object. Figure 3-1 shows the properties of a style object. In this figure, the geometric properties—those that apply primarily to geometric shapes—are highlighted. These properties are discussed in this chapter. The other style properties are shown in gray. These include

- the typographic style properties (those that apply primarily to typographic shapes), which are described in the chapter “Typographic Styles” in *Inside Macintosh: QuickDraw GX Typography*
- the style properties common to all objects, (owner count and tag list), which are described in the chapter “Style Objects” in *Inside Macintosh: QuickDraw GX Objects*

Figure 3-1 Style object with geometric properties highlighted



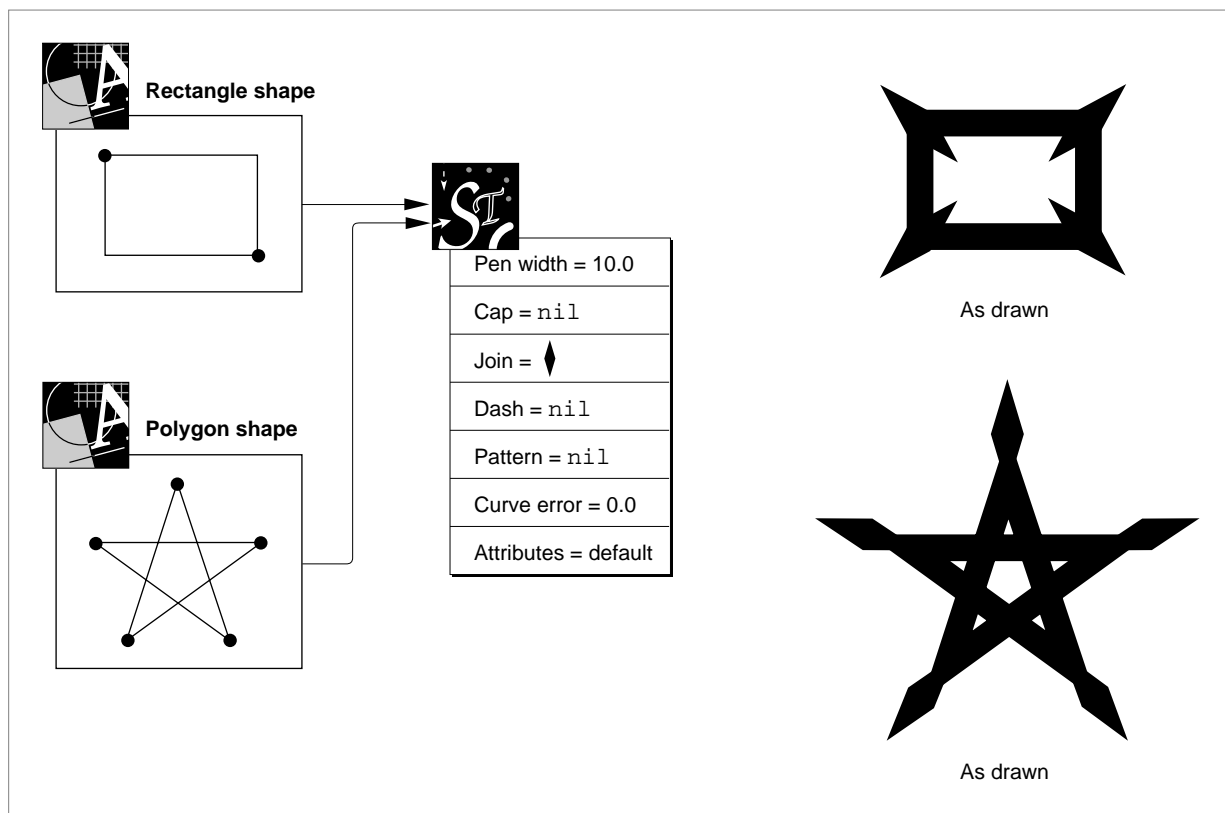
Style object

Pen width	Font	Run controls
Cap	Text face	Kerning adjustments array
Join	Text size	Glyph substitutions array
Dash	Alignment	Run features array
Pattern	Font variations	Priority justification override
Curve error	Encoding	Glyph justification overrides array
Attributes	Text attributes	
Owner count		
Tag list		

Geometric Styles

As Figure 3-2 depicts, a single style object can be shared by multiple shape objects.

Figure 3-2 Shared style objects



A geometric shape and a typographic shape can reference the same style object. The geometric shape uses the geometric style properties, which are described in this chapter, while the typographic shape uses the typographic style properties, which are described in the chapter “Typographic Styles” in *Inside Macintosh: QuickDraw GX Typography*.

Geometric Styles

QuickDraw GX typically handles style sharing for you. The section “Default Style Objects” on page 3-12 and the section “Associating Styles With Shapes” on page 3-36 describe the default style sharing behavior implemented by QuickDraw GX and how you can override this behavior.

As with all QuickDraw GX objects, a style object has an owner count, which reflects the number of existing references to the style object. When a new reference to a style object is created, the owner count of the style object is incremented; when a reference to a style object goes away, the owner count of the style object is decremented. When a style object has an owner count of 0, QuickDraw GX can free the memory used by the style object.

References to style objects typically include those contained in shape objects and those contained in variables in your application. QuickDraw GX manages the owner counts corresponding to references in shape objects for you; you are responsible for managing the owner counts corresponding to variables in your application. The chapter “Introduction to QuickDraw GX Objects” in *Inside Macintosh: QuickDraw GX Objects* explains owner counts and owner count management in more detail.

Incorporating Stylistic Variations Into Shape Geometries

When you draw a shape, QuickDraw GX applies the information in the style object of the shape to the shape’s geometry. For example, style objects contain a pen width property, described in full later in this chapter. When you draw a line shape, QuickDraw GX draws the line with the width specified in the pen width property of the style object associated with the line shape. As drawn, the thick line looks like a filled polygon. However, even after drawing the line shape, the shape still contains a line geometry.

QuickDraw GX provides a mechanism for incorporating the stylistic variations contained in a style object directly into the geometry of a shape object. This mechanism is the `GXPrimitiveShape` function, which is described in full in the next chapter, “Geometric Operations.”

If you make changes to a shape’s style object and then call the `GXPrimitiveShape` function, QuickDraw GX changes the shape’s shape type, shape fill, and shape geometry to incorporate the new stylistic variations. Basically, the same process that happens when drawing the shape happens directly to the shape’s geometry.

For example, Figure 3-3 shows a line shape. If you alter the style of this line shape to include a pen width of 10, the line shape effectively becomes a filled polygon shape.

If you were to apply the `GXPrimitiveShape` function to this thick line shape, the `GXPrimitiveShape` function would change the shape type to the polygon type, the shape fill to even-odd shape fill, and the shape geometry to a list of the four geometric points that define the polygon, as shown in Figure 3-3.

Another example, the result of applying the `GXPrimitiveShape` function to a framed rectangle with a thick pen width is shown in Figure 3-4. In this case, the result of the `GXPrimitiveShape` function is a filled polygon shape with two contours: an inside contour and an outside contour.

Figure 3-3 Effects of the `GXPrimitiveShape` function on a line shape

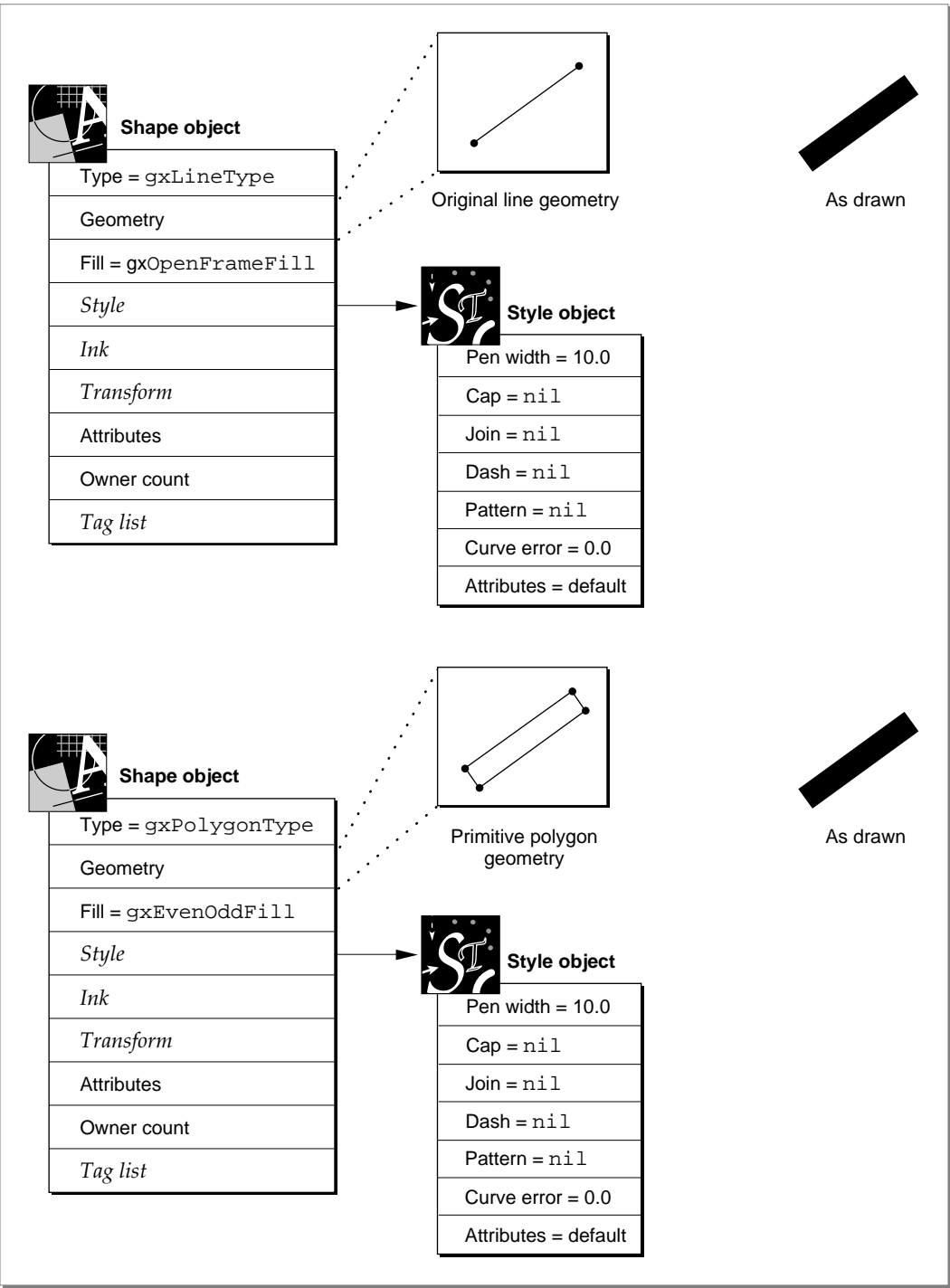
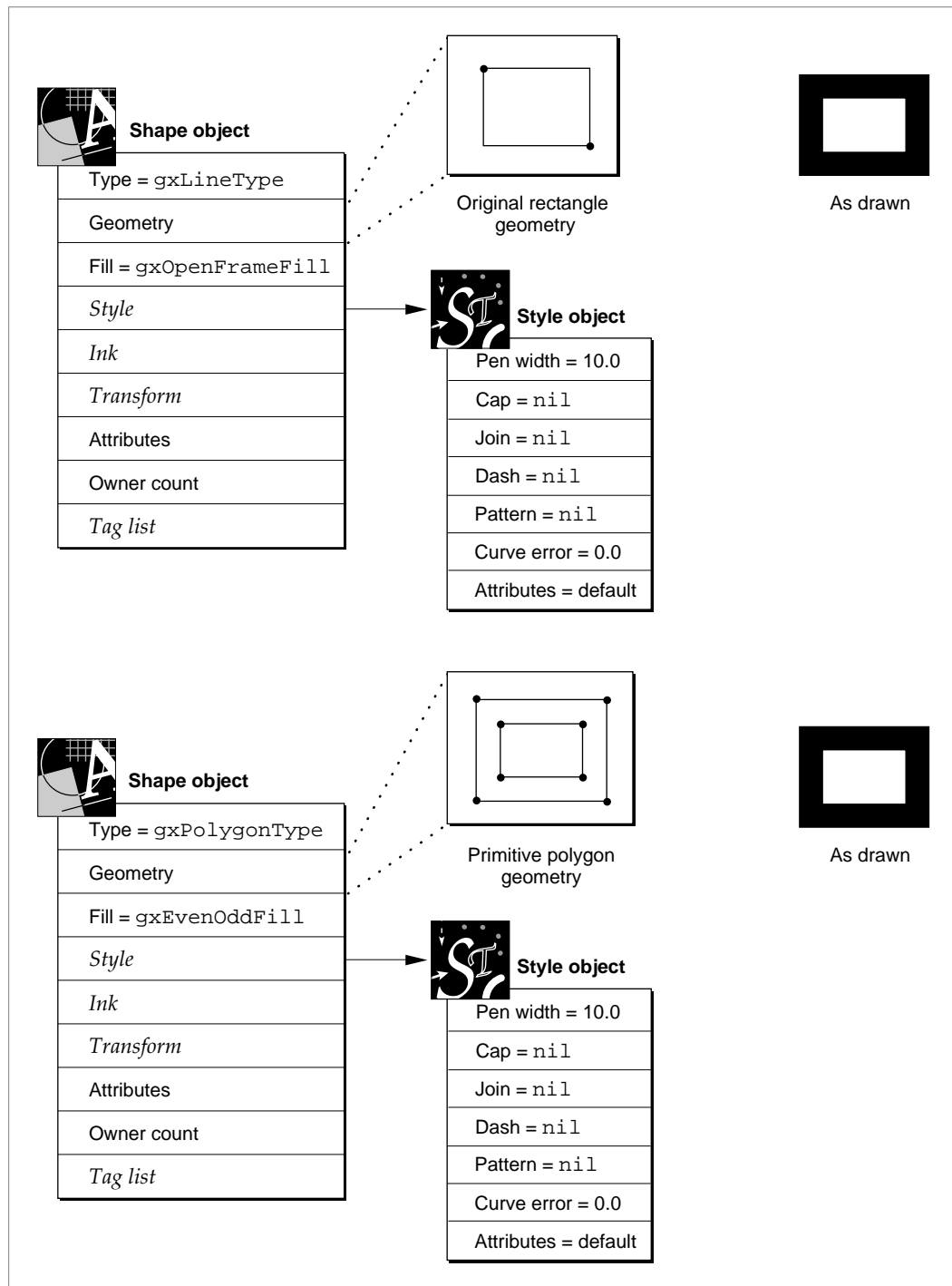


Figure 3-4 Effects of the `GXPrimitiveShape` function on a rectangle shape

Geometric Styles

Notice that the `GXPrimitiveShape` function does not affect the style object of the shape: it merely incorporates the existing style information into the geometry of the shape.

The result of calling the `GXPrimitiveShape` function is called a **primitive shape**, or a shape in its primitive form. Primitive shapes include

- empty shapes and full shapes, which are described in Chapter 3, “Geometric Shapes”
- filled rectangle, polygon, and path shapes, which are also described in Chapter 3, “Geometric Shapes”
- hairline framed shapes, which are described on page 3-16
- glyph shapes, which are described in *Inside Macintosh: QuickDraw GX Typography*
- bitmap shapes, which are described in Chapter 5, “Bitmap Shapes”

QuickDraw GX uses primitive shapes for caps, joins, dashes, and patterns, which are discussed throughout the rest of this chapter, and for clip shapes, which are discussed in the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Style Properties

Like most QuickDraw GX objects, each style object has an owner count and a list of tags. These properties are described in detail in the chapter “Introduction to QuickDraw GX Objects” in *Inside Macintosh: QuickDraw GX Objects*.

In addition to the owner count and the tag list, each style object contains properties that primarily affect the drawing of geometric shapes and properties that primarily affect the drawing of typographic shapes.

The style properties that primarily affect geometric shapes include the following:

- **Curve error.** This property specifies the allowable amount of error when QuickDraw GX converts a path shape into a polygon shape. It also specifies how far apart geometric points must be for QuickDraw GX to consider them separate points when simplifying or reducing a shape.
- **Pen width.** This property specifies the width of the pen QuickDraw GX uses to draw the contours of a shape.
- **Style attributes.** This property is a group of flags that allow you to specify how QuickDraw GX places the pen with respect to a shape’s geometry and whether the shape should be constrained to a grid when drawn.
- **Caps.** This property specifies what QuickDraw GX should draw at the start and the end of a shape’s contours. QuickDraw GX allows you to use any geometric shape (for example, a polygon shaped like an arrow head) as a start cap or end cap.
- **Join.** This property specifies what QuickDraw GX should draw at the corners of a shape’s contours. QuickDraw GX provides two standard join types (one for round corners and one for sharp corners), although QuickDraw GX allows you to specify any geometric shape as a join.

Geometric Styles

- **Dash.** This property specifies how QuickDraw GX should dash the contours of a shape. As with caps and joins, you can specify any geometric shape to dash the frame of another shape. However, you can also dash a shape with glyphs, which gives the effect of fitting text to a shape's frame.
- **Pattern.** This property specifies how QuickDraw GX should fill the geometry of a shape. You can use geometric shapes, glyphs shapes, or bitmap shapes as patterns.

The sections “Curve Error” on page 3-14, “The Geometric Pen” on page 3-15, “Style Attributes” on page 3-17, and “Interactions Between Caps, Joins, Dashes, and Patterns” on page 3-22 discuss these style properties in more detail.

The typographic style properties, which include font, text size, text face, and so on, are described in *Inside Macintosh: QuickDraw GX Typography*.

Default Style Objects

When you call the `GXNewStyle` function, which is described in the chapter “Style Objects” in *Inside Macintosh: QuickDraw GX Objects*, QuickDraw GX creates and returns a new style object. All of the new style object's properties are set to standard initial values. Once you have created a new style object, you can change the values of its properties, but you cannot change the behavior of the `GXNewStyle` function itself; it always returns a style object with these values for the geometric style properties:

- owner count: 1
- tag list: no tags
- style attributes: no attributes
- curve error: 0.0
- pen width: 0.0
- cap
 - cap attributes: no attributes
 - start cap: none
 - end cap: none
- join
 - join attributes: no attributes
 - join: none
 - join miter: Fixed1

Geometric Styles

- dash
 - dash attributes: no attributes
 - dash: none
 - dash advance: 0.0
 - dash phase: 0.0
 - dash scale: Fixed1
- pattern
 - pattern attributes: no attributes
 - pattern: none
 - pattern grid: (0.0,0.0), (0.0,0.0)

The chapter “Typographic Styles” in *Inside Macintosh: QuickDraw GX Typography* discusses the default style values for the typographic style properties.

Although you cannot change the behavior of the `GXNewStyle` function, QuickDraw GX provides another method for creating new style objects—a method that you can modify. When you create a new shape with the `GXNewShape` function, QuickDraw GX returns a copy of the default shape of the requested type. Since you can change the default shapes, you can also change the style objects that they reference.

Initially, all of the default shape objects reference the same style object. Whenever you create a new shape, it, too, references this style object. There are two ways in which you can change the style object associated with a new shape:

- You can call a function such as `GXSetShapePen`, which makes a copy of the style object specifically for your new shape before changing its pen width.
- You can obtain a reference to your new shape’s style object by calling the `GXGetShapeStyle` function, and then you can call a function such as `GXSetStylePen`, which does not make a copy of the style object. Instead, it affects the style object directly, which, in effect, changes the default style for all the default shapes.

By calling functions such as `GXSetShapePen` on each of the default shapes, you can create a different style object for each default shape. See the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects* for more information about default shapes.

Curve Error

Curve error is the only geometric style property that doesn't affect the drawing of a shape; instead, it affects the geometric points of the shape's geometry when performing geometric operations, shape type conversions, and shape simplifications. The **curve error** property determines how far away two points must be for QuickDraw GX to consider them as separate points in these cases:

- **Geometric operations.** QuickDraw GX guarantees that the results of the geometric operations described in the chapter “Geometric Operations” in this book, such as `GXIntersectShape` or `GXUnionShape`, have no two points closer than the value of the curve error of the target shape.
- **Insetting shapes.** A special case of geometric operation, the `GXInsetShape` function, which is described in the chapter “Geometric Operations” in this book, can produce results with an unusually large number of geometric points. Because the inset of a quadratic Bézier curve is not a quadratic Bézier curve itself, multiple insets of tight curve shapes can cause the number of geometric points to grow dramatically. As with the other geometric operations, the result of the `GXInsetShape` function has no two consecutive points closer than the value of the curve error of the target shape.
- **Path to polygon conversions.** The curve error also determines the maximum error when converting a path shape to a polygon (for example, with the code `GXSetShapeType (aPathShape, gxPolygonType)`). The distance between the original path and the resulting polygon is always less than the value of the curve error. If the curve error is 0, QuickDraw GX performs the path to polygon conversion simply by removing all off-curve control points, which gives a fairly rough approximation.
- **Shape simplifications.** The functions `GXReduceShape` and `GXSimplifyShape`, which are described in more detail the chapter “Geometric Operations” in this book, perform a number of simplifications on shapes (for example, removing geometric points unnecessary to the geometry and unwinding crossed contours). In addition to their other simplifications, these functions remove all consecutive (on-curve) geometric points within a distance of less than the curve error.

The sections “Using Curve Error When Converting Paths to Polygons” on page 3-45 and “Using Curve Error When Reducing Shapes” on page 3-49 give examples of using curve error, and the section “Getting and Setting Curve Error” on page 3-114 describes the functions you can use to manipulate this style property.

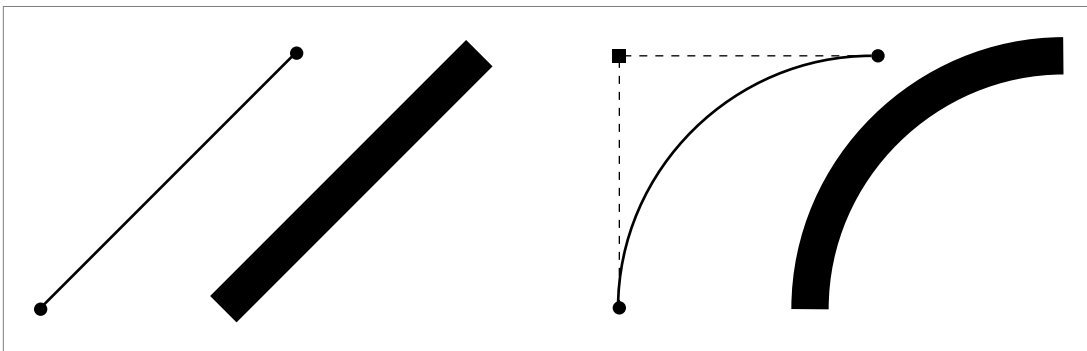
The Geometric Pen

The contours of framed geometric shapes are drawn with the QuickDraw GX **geometric pen**. You can specify the width of this pen using the pen width property of the style object, and you can specify where to place the pen relative to the contours of a shape using the style attributes, which are described in “Style Attributes” beginning on page 3-17.

Conceptually, the QuickDraw GX geometric pen is a line that QuickDraw GX drags along the contours of the shape being drawn—always keeping it perpendicular to the contours. In effect, the geometric pen turns a framed geometry into a filled one. For example, a line shape, which is always framed, becomes the equivalent of a filled polygon after QuickDraw GX applies the geometric pen.

Figure 3-5 shows the effect of the geometric pen. This figure shows two geometries—a line geometry and a curve geometry—and how QuickDraw GX draws them with a pen width of 15.

Figure 3-5 The QuickDraw GX geometric pen

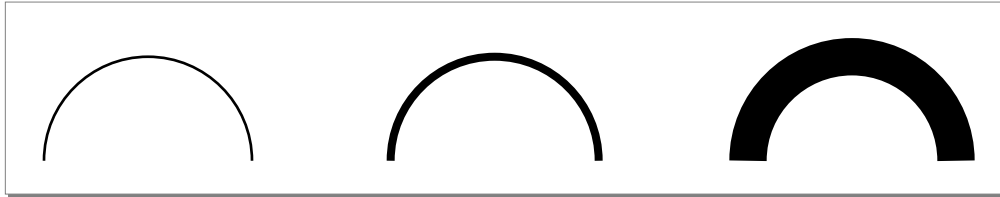


Notice that the ends of the thick line contour and the thick curve contour in Figure 3-5 are perpendicular to the direction of the contours themselves.

Geometric Styles

Figure 3-6 shows the effect of different pen widths on a semicircular path shape.

Figure 3-6 Differing pen widths



Setting a value of 0 for the pen width property has special meaning. Instead of indicating an infinitely thin pen, it indicates that a shape's contours should be drawn using **hairlines**—the thinnest line renderable on the device to which the shape is drawn. A hairline is always one pixel wide and is always centered about the shape's geometry.

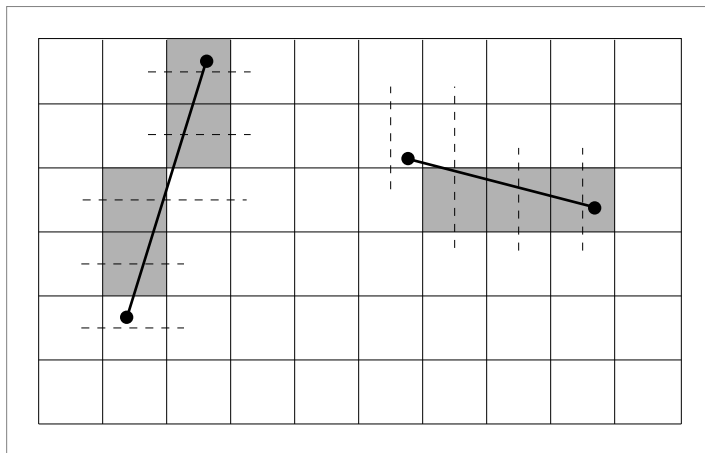
One important use of hairlines is to make point shapes visible. QuickDraw GX draws point shapes under only two conditions: if the pen width is 0, indicating a hairline point, in which case exactly one pixel is drawn, or if the point has a start cap, which is described in “Caps” beginning on page 3-23.

When drawing a hairline, QuickDraw GX uses this algorithm to determine which pixels to include:

- If the contour being drawn is more vertical than horizontal, QuickDraw GX includes a pixel if the contour crosses the horizontal center line of the pixel.
- If the contour being drawn is more horizontal than vertical, QuickDraw GX includes a pixel if the contour crosses the vertical center line of the pixel.

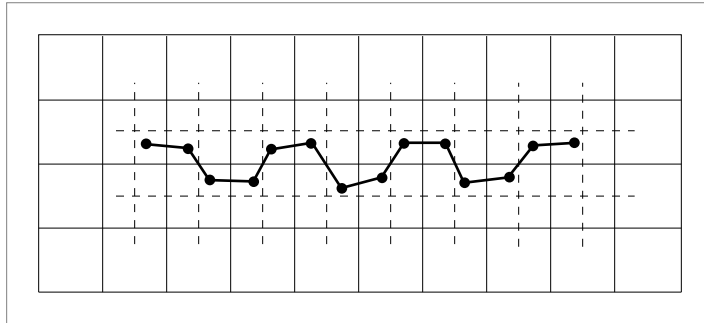
Figure 3-7 depicts this algorithm.

Figure 3-7 Pixels included in a hairline



In extreme cases, this algorithm can cause no pixels to draw, as shown in Figure 3-8.

Figure 3-8 A geometry with no hairline



The section “Manipulating Pen Width and Placement” on page 3-51 gives an example of using the pen width property, and the section “Getting and Setting the Pen Width” on page 3-119 describes the functions you can use to manipulate it.

Style Attributes

The **style attributes** property of a style object contains six attributes that affect the drawing of a shape. Four of these attributes affect how QuickDraw GX places the geometric pen relative to the contours of a shape:

- The center-frame style attribute, which is the default, indicates that the QuickDraw GX should center the geometric pen along the shape’s contours.
- The inside-frame style attribute indicates that QuickDraw GX should position the pen along the inside of a shape’s contours.
- The outside-frame style attribute indicates that QuickDraw GX should position the pen along the outside of shape’s contours.
- The auto-inset style attribute affects the definition of the inside and outside of a contour.

These four attributes are discussed in the next section, “Pen Placement.”

There are also two style attributes that determine whether the geometric points of a shape are constrained to a grid when the shape is drawn:

- The source-grid style attribute constrains the geometric points of a shape to integer values before applying the shape’s style and transform information.
- The device-grid style attribute constrains the geometric points of a shape to integer pixel positions after applying the shape’s style and transform information.

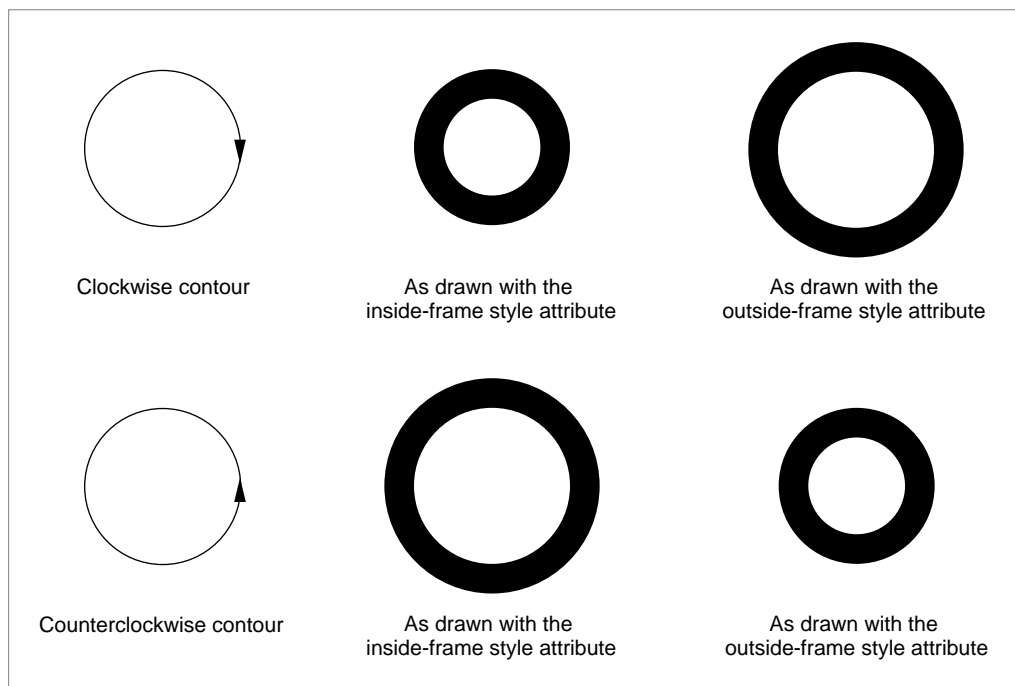
These two attributes are discussed in the section “Grids” beginning on page 3-20.

Pen Placement

You can use the center-frame, inside-frame, and outside-frame style attributes to specify where QuickDraw GX should position the pen with respect to the shape's geometry. QuickDraw uses these attributes to position the pen, which also affects the placement of dashes and how dashes are clipped. For some examples, see "Insetting Dashes" beginning on page 3-73 and "Combining Caps, Joins, Dashes, and Patterns" beginning on page 3-91.

Figure 3-9 shows the results of these style attributes. Notice that QuickDraw GX considers contour direction when determining which side of a contour is the inside: the right side of the contour is the inside, while the left side of the contour is the outside.

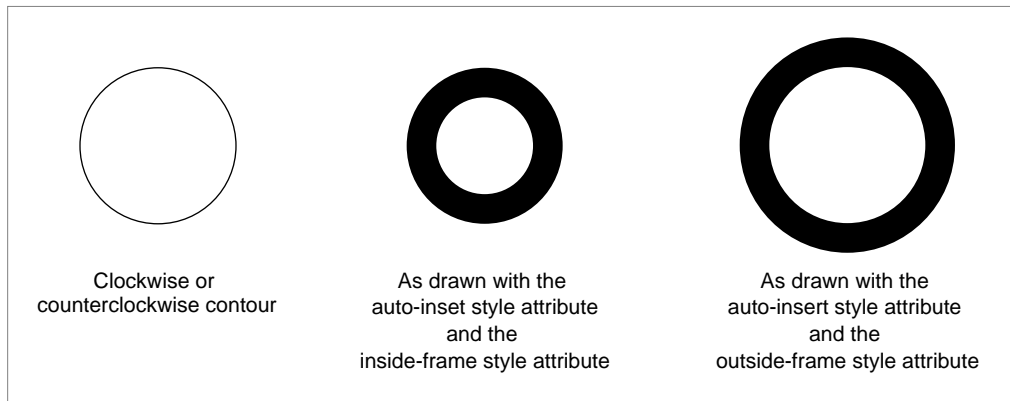
Figure 3-9 Pen placement



Geometric Styles

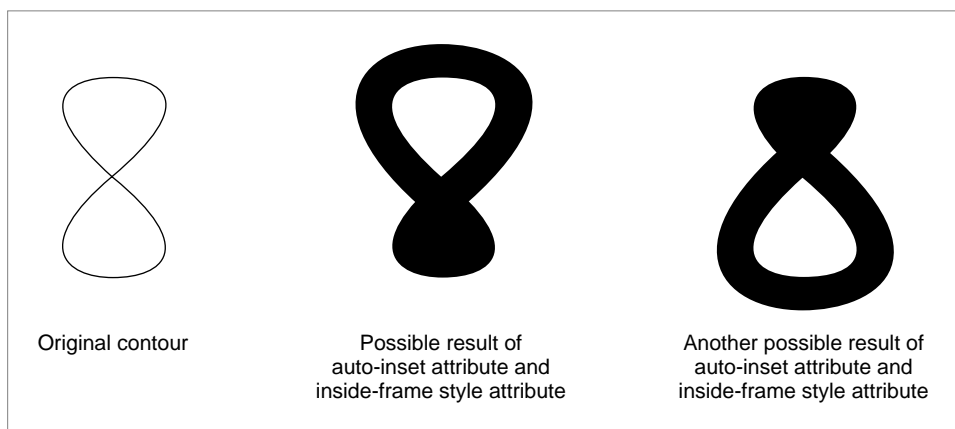
QuickDraw GX also provides the auto-inset style attribute, which allows you to specify that QuickDraw GX should ignore contour direction when determining which side of a contour is the inside. When you set this style attribute, QuickDraw GX determines the true inside of a contour, rather than using the right side as the inside. Figure 3-10 shows the effect of setting the auto-inset style attribute for the shapes depicted in Figure 3-9.

Figure 3-10 Effect of the auto-inset style attribute



When a contour crosses over itself, the results of setting the auto-inset style attribute are unpredictable, as the contour has no true inside (or, actually, has multiple true insides). For the figure-eight shape in Figure 3-11, setting the `gxAutoInsetStyle` and the `gxInsideFrameStyle` style attributes could lead to one of two results.

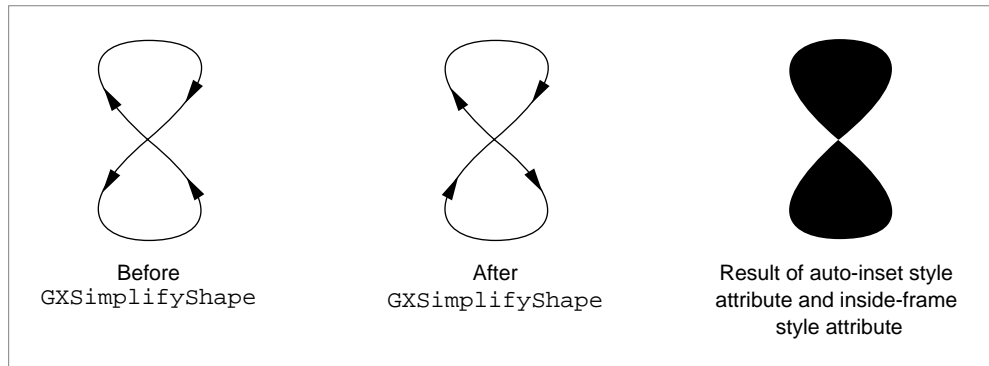
Figure 3-11 Effect of the auto-inset and inside-frame style attributes for a crossed contour



Geometric Styles

To ensure that setting the auto-inset style attribute behaves as you would expect, you need to call the `GXSimplifyShape` function, which is described in the chapter “Geometric Operations” in this book. This function redefines the shape’s geometry to eliminate crossed contours, as shown in Figure 3-12.

Figure 3-12 Eliminating crossed contours



The section “Manipulating Pen Width and Placement” on page 3-51 gives an example of specifying pen placement. The section “Style Attributes” on page 3-98 defines the style attributes enumeration, and the section “Getting and Setting Style Attributes” on page 3-109 describes the functions you can use to manipulate them.

Grids

From the initial geometry specification to the final image rendering, each QuickDraw GX shape exists in a number of different coordinate spaces. You describe a shape’s geometry in geometry space, the style and transform modifications happen in local space, the shape then exists in one or more view ports’ global spaces, and the shape is finally rendered in the pixels of a view device’s device space.

In each of these coordinate spaces, QuickDraw GX allows fractional coordinate values. When you specify points in a shape’s geometry, you are not limited to integer values, such as (1, 1) or (–10, 10). Instead, you can specify that the shape’s geometric points fall between integral positions in the geometry space’s coordinate grid, for example (0.5, 0.5). During each transformation of the shape from geometry to rendering, QuickDraw GX maintains fractional coordinate values.

The style attributes property of a style object contains two flags that allow you to suppress fractional coordinate values—that is, these flags allow you to constrain a shape’s geometric points to integer coordinate values in the different coordinate systems.

Geometric Styles

The source-grid style attribute indicates that QuickDraw GX should constrain the shape's geometric points to integral positions on the local space grid, before making the style and transform modifications.

The device-grid style attribute indicates that QuickDraw GX should constrain the shape's geometric points to integral positions (that is, pixel positions) on the device space grid, after making style, transform, and view port modifications.

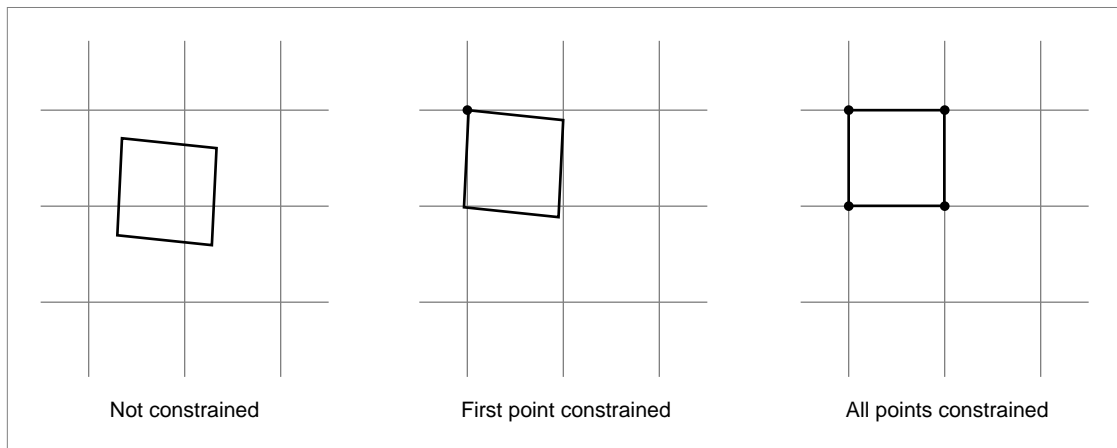
Note

These style attributes only affect a shape while it is being drawn. They do not affect the geometric points you specify in the original shape geometry. ♦

To constrain a shape to integral positions on a coordinate space's grid, QuickDraw GX moves the entire shape (that is, all the shape's geometric points) so that the shape's first geometric point lies on the nearest grid position, and then moves each remaining geometric point to the nearest grid position.

Figure 3-13 depicts the grid-constraining algorithm.

Figure 3-13 Constraining shapes to grids



The sections “Constraining Shape Geometries to Grids” on page 3-40 and “Constraining Shapes to Device Grids” on page 3-42 give examples of the grid-constraining style attributes. The section “Style Attributes” on page 3-98 defines the style attributes enumeration and the section “Getting and Setting Style Attributes” beginning on page 3-109 describes the functions you can use to manipulate them.

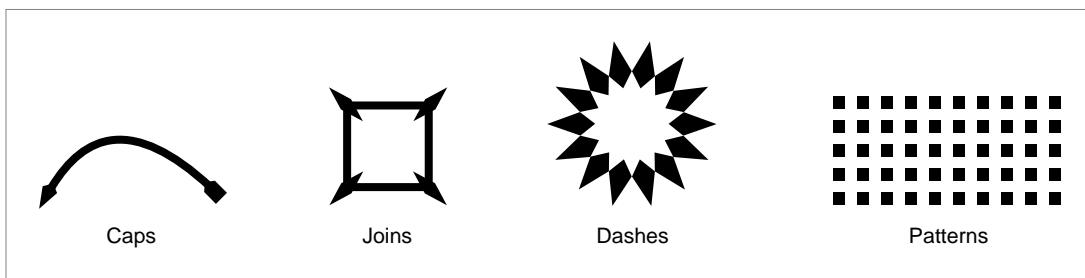
Interactions Between Caps, Joins, Dashes, and Patterns

The cap, join, dash, and pattern properties of the style object allow you to change the way QuickDraw GX draws the contours of a shape. The cap and join properties allow you to place arbitrary shapes on the geometric points of a shape's contours. For example, you can place arrow heads at the ends of a line, or you can put rounded edges at the corners of a rectangle. The dash property allows you to dash the contours of one shape with another shape. For example, you could dash a line with a circular path shape to get a dotted line.

The pattern property allows you to fill a shape (or the frame of a shape drawn with a thick pen width) with a repeated pattern of another shape. For example, you could fill a large square shape with a pattern of small squares to get a checkerboard.

Figure 3-14 shows some of the stylistic variations possible with caps, joins, dashes, and patterns.

Figure 3-14 Caps, joins, dashes, and patterns



There is one important rule that applies to all four of these properties: Cap shapes, join shapes, dash shapes, and pattern shapes must all be in their primitive form. When QuickDraw GX uses a cap, join, dash, or pattern shape, it ignores the stylistic variations of that shape. If you want a cap, join, dash, or pattern shape to have stylistic variations itself, you must first incorporate those stylistic variations into the shape using the `GXPrimitiveShape` function.

As an example, specifying a line shape with a thick pen (like the one in Figure 3-3 on page 3-9) as a cap, join, dash, or pattern shape may produce an unexpected result or post an error, since the shape is not in its primitive form. However, if you use the `GXPrimitiveShape` function, you can convert the line to a filled polygon, which is a perfectly acceptable cap, join, dash, or pattern shape.

The sections “The Cap Structure” on page 3-99, “The Join Structure” on page 3-101, “The Dash Structure” on page 3-103, and “The Pattern Structure” on page 3-106 discuss the types of shapes appropriate to use as caps, joins, dashes, and patterns.

Geometric Styles

As another example, a polygon with zero contours is not an acceptable cap, join, dash, or pattern shape, as it is not in its primitive form. Similarly, any shape with the no-fill shape fill is not in its primitive form. However, the empty shape, which is in its primitive form, is an acceptable cap, join, dash, or pattern shape. You can find more information about polygon shapes, polygon contours, and empty shapes in Chapter 2, “Geometric Shapes,” in this book.

You can always be sure your cap, join, dash, or pattern shape is in the correct form by calling the `GXPrimitiveShape` function, which is described in Chapter 4, “Geometric Operations,” before setting the corresponding style property.

As for typographic shapes, text and layout shapes are not in their primitive form, but glyph shapes are acceptable as cap, join, dash, and pattern shapes so long as they have no text face or tags and do not have caps, joins, dashes, or patterns themselves.

For more information, see *Inside Macintosh: QuickDraw GX Typography*.

You can use bitmap shapes as patterns, but not as caps, joins, or dashes, and you cannot use picture shapes for caps, joins, dashes, or patterns.

The next few sections describe caps, joins, dashes, and patterns in more detail.

Caps

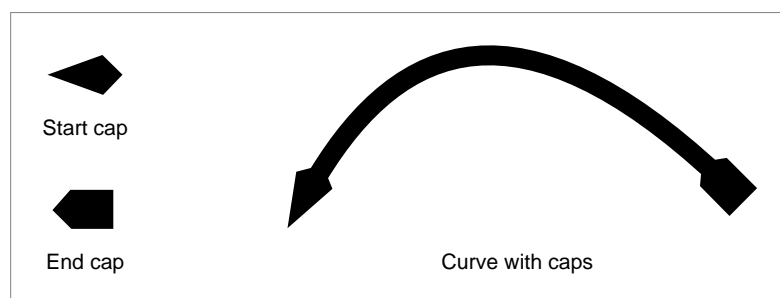
QuickDraw GX allows you to specify **cap shapes**—what to draw at the start and at the end of a shape’s contours. In particular, you can specify a start cap for any point shape, and you can specify a start cap and an end cap for any line, curve, polygon, or path shape that has open-frame shape fill.

In fact, the only way to draw a point shape is to specify a start cap for it (unless its style object has a pen width property with a value of 0, in which case QuickDraw GX draws the point shape as a single pixel).

QuickDraw GX uses the **cap property** of a shape’s style object to store information about the start cap and end cap of the shape.

Figure 3-15 shows how QuickDraw GX adds a cap to a contour by centering the cap shape at the end of the contour, scaling the cap shape by the pen width, and rotating the cap shape to match the slope of the contour.

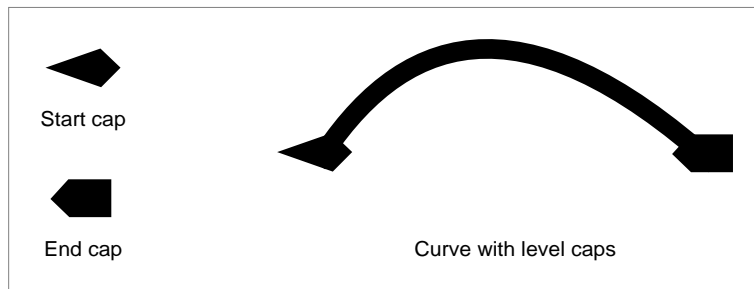
Figure 3-15 A shape with caps



Geometric Styles

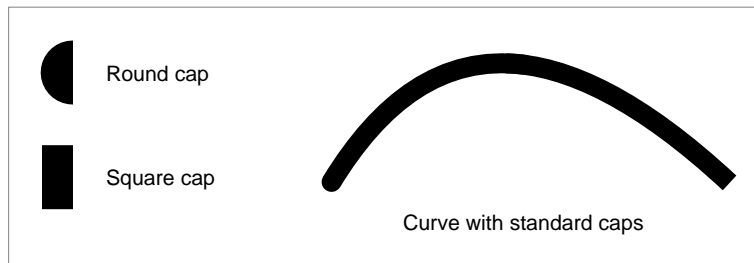
The cap property of a style object includes a **cap attributes** field, which allows you to specify **level caps**—caps that QuickDraw GX does not rotate to match the slope of the contour—as shown in Figure 3-16.

Figure 3-16 A shape with level caps



You can create two standard cap types by specifying half a square or a semicircle for the start cap or end cap shape, as shown in Figure 3-17.

Figure 3-17 Standard cap shapes



The sections “Adding Caps to a Shape” on page 3-57 and “Adding Standard Caps to a Shape” on page 3-59 give examples of using the cap property, the sections “The Cap Structure” on page 3-99 and “Cap Attributes” on page 3-101 describe the data structures used to store information about caps, and the section “Getting and Setting Caps” beginning on page 3-123 describes the functions you can use to manipulate caps.

Joins

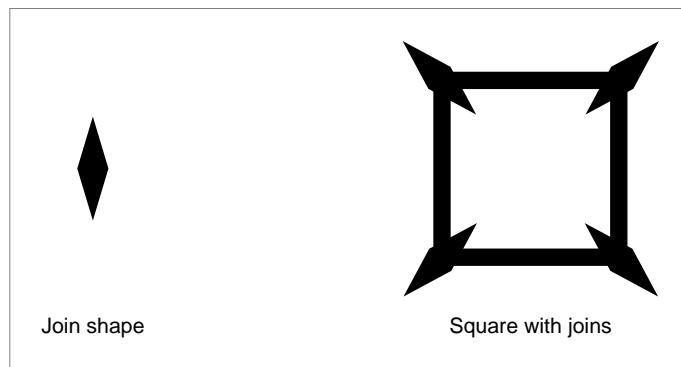
QuickDraw GX allows you to specify a **join shape** to be drawn at the corners of another shape's contours. In particular, you can specify a join shape for any rectangle, polygon, or path shape that has an open-frame shape fill or a closed-frame shape fill:

- For shapes with the closed-frame shape fill, QuickDraw GX draws the specified join shape at every on-curve geometric point of each contour.
- For shapes with the open-frame shape fill, QuickDraw GX draws the specified join shape at every on-curve geometric point between the first point and the last point of each contour.

QuickDraw GX uses the **join property** of a shape's style object to store information about the joins of a shape.

Figure 3-18 shows how QuickDraw GX adds a join to a contour by centering the join shape on the on-curve geometric points, scaling the join shape by the pen width, and rotating the join shape to match the mid-angle of the two line segments that make up the corner.

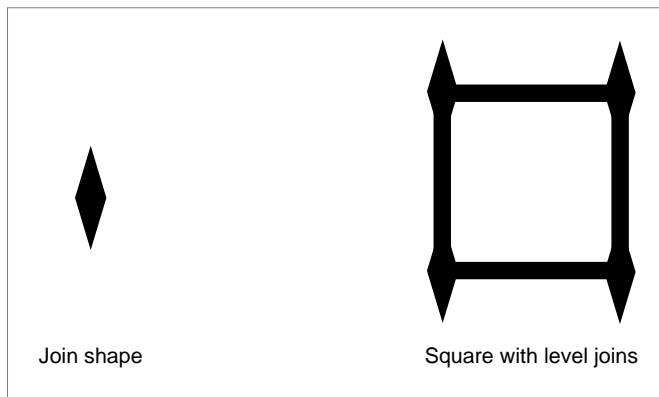
Figure 3-18 A shape with joins



Geometric Styles

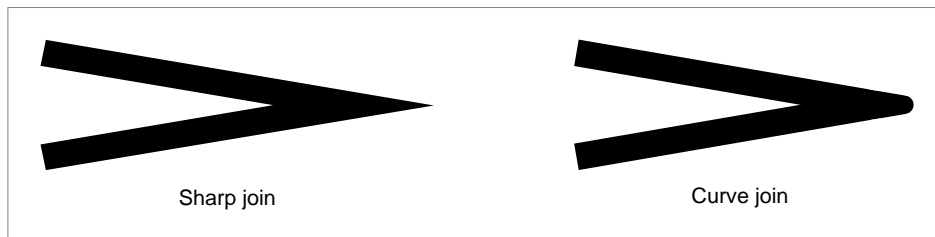
The join property of a style object includes a **join attributes** field, which allows you to specify **level joins**—joins that QuickDraw GX does not rotate to match the slope of the contour—as shown in Figure 3-19.

Figure 3-19 A shape with level joins



You can also use the join attributes to specify two types of standard joins—sharp joins and curve joins, as shown in Figure 3-20.

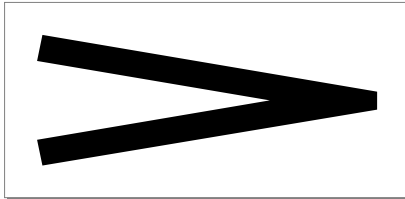
Figure 3-20 Standard joins



Geometric Styles

For sharp joins, QuickDraw GX allows you to specify a **miter**—the maximum distance between the actual corner of a shape’s geometry and the corner as drawn, as shown in Figure 3-21.

Figure 3-21 Sharp join with miter



The sections “Adding Joins to a Shape” on page 3-61 and “Adding Standard Joins to a Shape” on page 3-64 give examples of using the join property, the section “The Join Structure” on page 3-101 and “Join Attributes” on page 3-102 describe the data structures used to store information about joins, and the section “Getting and Setting Joins” on page 3-129 describes the functions you can use to manipulate them.

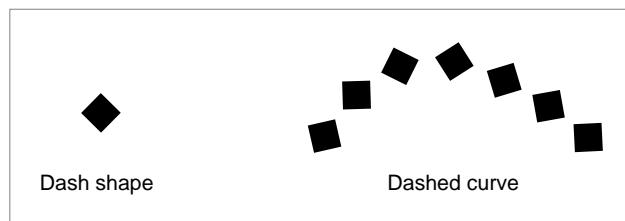
Dashes

With QuickDraw GX, you can specify that framed shapes should be drawn with dashed, instead of solid, contours. In particular, you may specify a **dash shape** for any line, curve, rectangle, polygon, or path shape that has an open-frame shape fill or a closed-frame shape fill.

QuickDraw GX uses the **dash property** of a shape’s style object to store information about how to dash the shape.

Figure 3-22 shows how QuickDraw GX dashes a contour by placing copies of the dash shape along the contour at regular intervals, and rotating the dash shape to match the slope of the contour.

Figure 3-22 A dashed shape



Geometric Styles

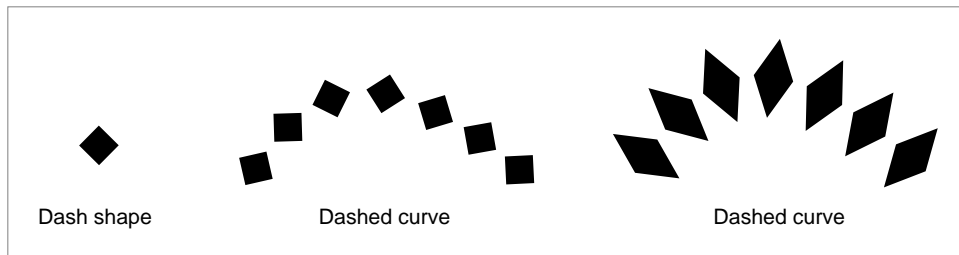
When drawing a dashed shape, QuickDraw GX automatically scales the dash shape up by the pen width of the dashed shape. However, unlike cap and joins, QuickDraw GX scales dashes only *perpendicularly to the dashed contour*.

For example, if the height of the dash shape is 1.0, then QuickDraw GX draws the dashes with a height equal to the dashed shape's pen width. If the height of the dash shape is 2.0, then QuickDraw GX draws the dashes with a height equal to twice the dashed shape's pen width.

Since the dash shape is scaled up by the pen width of the dashed shape, QuickDraw GX provides a way for you to scale the dash down, as well, by providing a scaling factor, called the **dash scale**, in one of the fields of the dash structure. QuickDraw GX multiplies the height of the dash by the pen width and then divides by the dash scale.

Figure 3-23 shows the effect of different pen widths on the same dash shape. In this example, the dash shape has height of 10.0 (its y-coordinates span from -5.0 to 5.0). The shape being dashed is a curve. The curve is shown first with a pen width of 10.0 and a dash scale of 10.0, which keeps the dimensions of the dash shape unchanged. The curve is then shown with a pen width of 20.0 and a dash scale of 10.0, which doubles the size of the dash shape in the y-coordinate direction.

Figure 3-23 Scaling a dash shape

**Note**

Glyph shapes are an exception to this scaling rule. If the dash shape is a glyph shape, QuickDraw GX does not scale the dashes (which in this case would be glyphs) to the dashed shape's pen width. ♦

Geometric Styles

Notice that the position of a dash shape in the coordinates of its geometry space is significant. For example, if the y-coordinates of the geometry of a dash shape span from 1.0 to 2.0, then QuickDraw GX draws the dashes at a distance of one pen width to the outside of the dashed contour (if the dash scale is 1.0). If the lowest y-coordinate of a dash shape is 2.0, then QuickDraw GX draws the dashes at a distance of two pen widths to the outside of the contour (if the dash scale is 1.0).

If the y-coordinates of the geometry of a dash shape are large enough and the scaling factor you provided in the dash structure is small enough, the dashes may exceed the pen width of the dashed shape. QuickDraw GX provides the clip dash attribute to indicate that QuickDraw GX should clip the dashes to the pen width, as illustrated in Figure 3-24.

Figure 3-24 Effect of the clip dash attribute

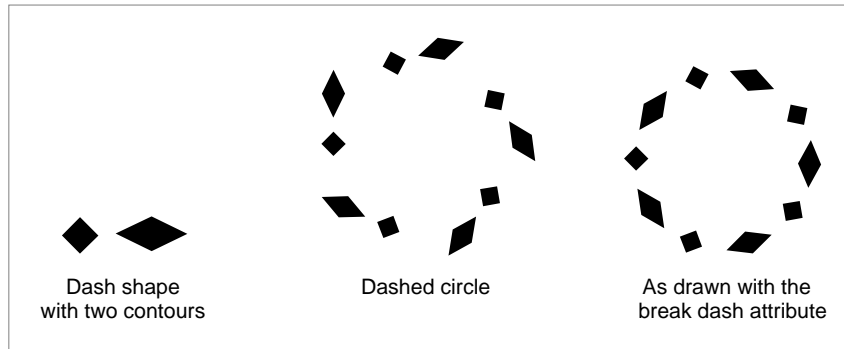


Setting the clip dash attribute causes some intricate interactions among dashes, caps, joins, and patterns. See “Interactions Between Caps, Joins, Dashes, and Patterns” on page 3-33 for more information.

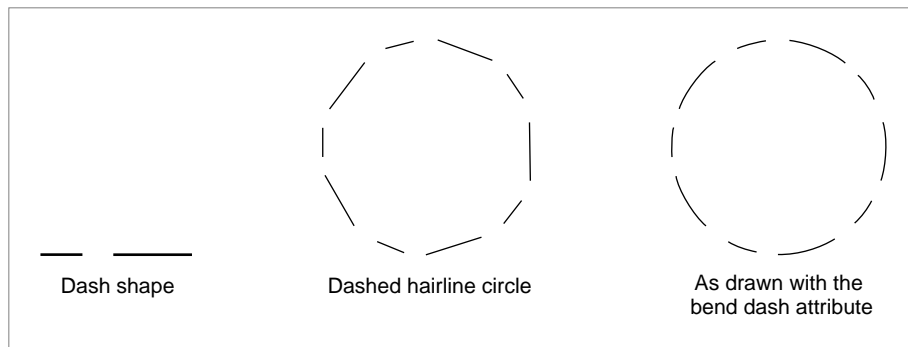
QuickDraw GX also allows you to control how far apart the dashes appear from one another, which is called the **dash advance**, and how far into the dash shape the dashing should start, which is called the **dash phase**.

The dash advance is the distance between the start of one dash shape and the start of the next dash shape along the contour. The dash phase indicates where the first dash should fall on a contour; it is a percentage of the dash advance.

When a dash shape has multiple contours, it is possible for the dashes not to fall on the contours of the dashed shape. For this situation, QuickDraw GX provides the break dash attribute, which indicates that each contour of the dash should be rotated and placed separately on the dashed shape’s contours. Figure 3-25 depicts the result of setting this dash attribute.

Figure 3-25 Effects of breaking a dash

Finally, QuickDraw GX provides a dashing feature that works only when dashing hairline contours. In this case, you can set the bend dash attribute, which indicates that QuickDraw GX should wrap the dash to fit the dashed contour exactly, as shown in Figure 3-26.

Figure 3-26 Effects of bending a dash

Geometric Styles

The following sections give examples of dashing:

- “Dashing a Shape” on page 3-66
- “Adjusting Dashes to Fit Contours” on page 3-70
- “Insetting Dashes” on page 3-73
- “Breaking and Bending Dashes” on page 3-74
- “Wrapping Text to a Contour” on page 3-80
- “Determining Dash Positions” on page 3-81

The section “The Dash Structure” on page 3-103 and “Dash Attributes” on page 3-105 describe the data structures used to store and communicate information about dashes, and the section “Getting and Setting Dashes” on page 3-134 describes the functions you can use to manipulate the dash property.

Patterns

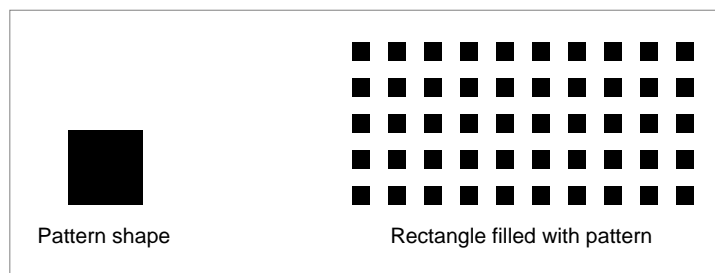
With QuickDraw GX, you can specify that certain shapes be filled with a pattern. For shapes with solid shape fills, QuickDraw GX fills the shape by repeating a pattern shape that you specify.

You can also pattern framed shapes. For example, if you pattern a rectangle shape with a closed-frame shape fill and a pen width of 20.0, QuickDraw GX would fill the frame of the rectangle with the pattern. See the section “Interactions Between Caps, Joins, Dashes, and Patterns” on page 3-33 for more intricate examples.

QuickDraw GX uses the **pattern property** of a shape’s style object to store information about how to pattern the shape.

Figure 3-27 shows how QuickDraw GX patterns a shape by filling the shape with copies of another shape, called the **pattern shape**, placed according to a regular grid that you specify.

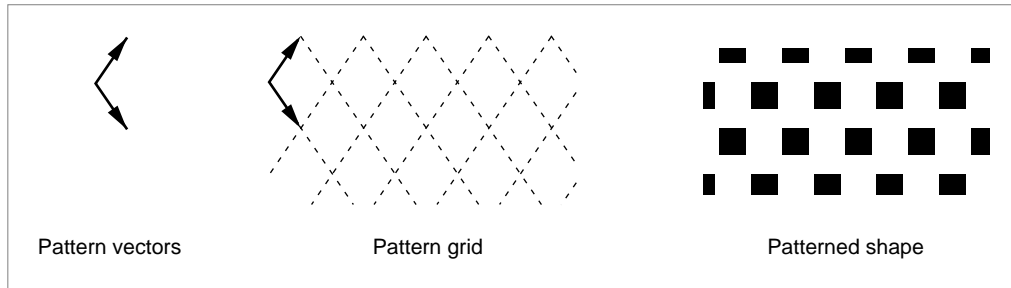
Figure 3-27 A shape with a pattern



Geometric Styles

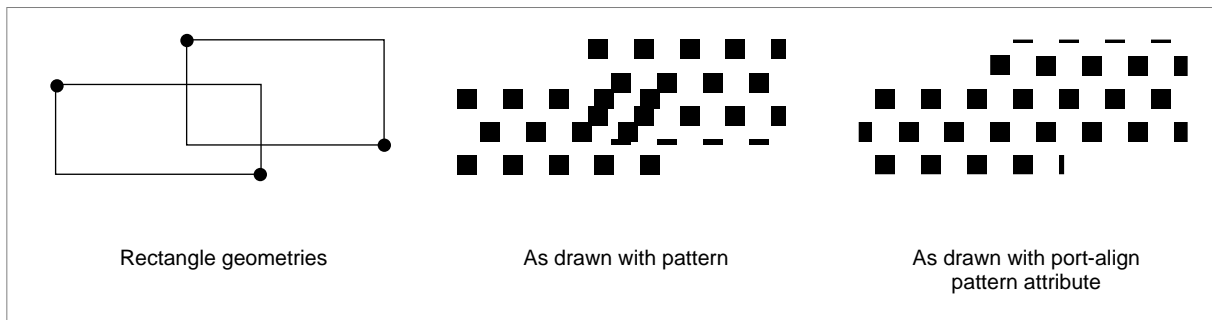
In addition to specifying the shape to use as the pattern shape, you also specify the **pattern grid** on which to place the pattern, as shown in Figure 3-28.

Figure 3-28 Pattern placed on a nonrectilinear grid



In addition, QuickDraw GX provides you with two **pattern attributes**: the port-align pattern attribute and the port-map pattern attribute. Setting the port-align pattern attribute allows you to specify that QuickDraw GX should align the pattern with the view device instead of with the geometry of the patterned shape. Figure 3-29 shows the effect of setting this attribute.

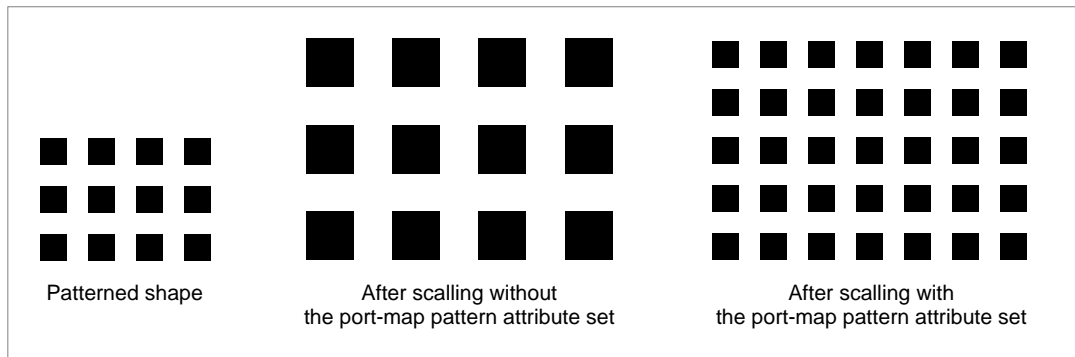
Figure 3-29 Effects of the port-align pattern attribute



Geometric Styles

The port-map pattern attribute indicates that the pattern shape should not be affected by transformations to the patterned shape. For example, if you set this pattern attribute, scaling the patterned shape by a factor of two does not also scale the pattern shape by a factor of two; instead, more of the pattern is shown. Figure 3-30 shows the effect of setting this attribute.

Figure 3-30 Effects of the port-map pattern attribute



The sections “Adding a Pattern to a Shape” on page 3-86 and “Determining Pattern Positions” on page 3-88 give examples of using patterns. The section “The Pattern Structure” on page 3-106 and “Pattern Attributes” on page 3-107 describes the data structures used to store information about patterns, and the section “Getting and Setting Patterns” on page 3-142 describes the functions you can use to manipulate patterns.

Interactions Between Caps, Joins, Dashes, and Patterns

The previous four sections show the results of adding a cap, a join, a dash, or a pattern to a QuickDraw GX shape. This section discusses how these stylistic variations interact when you add more than one of them at a time to the same shape.

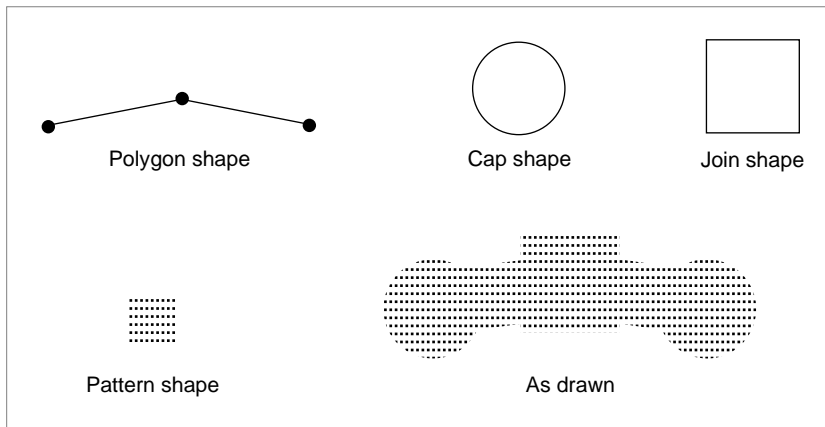
In general, these elements interact differently in each of these three cases:

- the shape does not have a dash but has one or more of the other three stylistic variations
- the shape does have a dash but the clip dash attribute is not set
- the shape does have a dash and the clip dash attribute is set

Geometric Styles

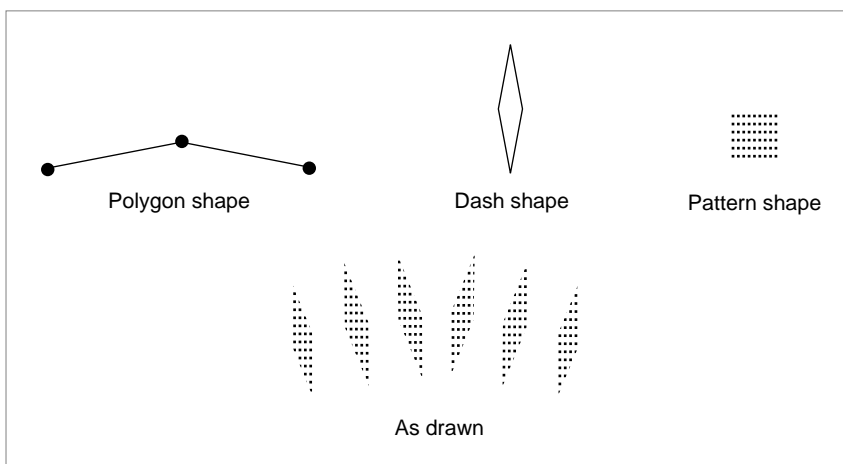
When a shape has a cap and a join, QuickDraw GX adds the caps to the beginnings and ends of the shape's contours, and adds the joins to the other on-curve geometric points of the shape's contours. If the shape also has a pattern, QuickDraw GX draws this pattern throughout the shape's frame as well as the shape's caps and joins, as shown in Figure 3-31.

Figure 3-31 A shape with a cap, join, and pattern



If a shape has a dash, but the clip dash attribute is not set, QuickDraw GX *ignores the caps and joins of the shape*. However, if the shape has a pattern, QuickDraw GX does draw the pattern throughout the dashes, as shown in Figure 3-32.

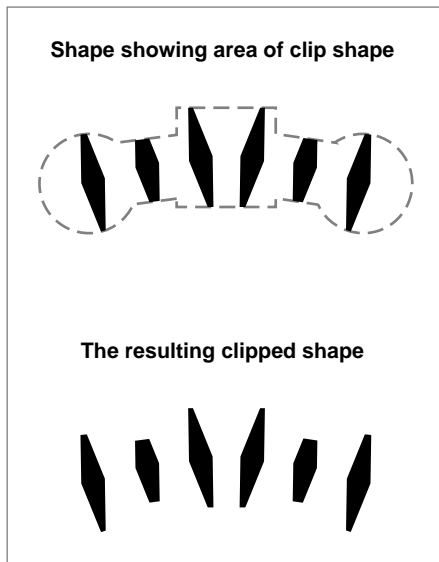
Figure 3-32 A shape with a dash and a pattern



Geometric Styles

Finally, if the shape has a dash and the clip dash attribute is set, QuickDraw GX does not ignore the caps and joins. Instead, the cap shapes and the join shapes are added to the clip shape that QuickDraw GX uses to clip the dashes. Patterns are not allowed in this case. Figure 3-33 shows the interaction of a cap, join, and clipped dash.

Figure 3-33 A shape with a clipped dash and a cap and join



The section “Combining Caps, Joins, Dashes, and Patterns” beginning on page 3-91 give examples of the interactions between caps, joins, dashes, and patterns.

Using Geometric Styles

This section shows you how to use styles to add stylistic variations to geometric shapes. In particular, this section show you how to

- create a style object, alter its properties, and associate the style with a shape
- alter the properties of a style object already associated with a shape
- constrain shapes to grids
- use curve error when approximating paths with polygons and when reducing shapes
- manipulate pen width and placement

Geometric Styles

- add caps to a shape, including round and square caps
- add joins to a shape, including standard round and sharp joins
- dash a shape
- adjust dashes to fit contours
- bend and break dashes
- wrap text by using glyphs as a dash shape
- determine dash positions
- add a pattern to a shape and determine pattern positions
- combine caps, joins, dashes, and patterns

Associating Styles With Shapes

QuickDraw GX provides two basic methods of altering stylistic information for shapes:

- using functions that operate on style objects directly
- using functions that operate on style objects indirectly through shape objects

The first category of functions require you to provide a reference to a style object, which you can obtain by using the `GXNewStyle` function to create a new style object, or by using the `GXGetShapeStyle` function to obtain a reference to an existing style object. (The `GXNewStyle` and `GXGetShapeStyle` functions are described in *Inside Macintosh: QuickDraw GX Objects*.)

Once you have a reference to a style object, you can use this category of functions to manipulate the style's properties; for example, you can use the `GXSetStylePen` function to change the pen width of the style.

If you obtained the reference to the style object using the `GXGetShapeStyle` function, then the style is already associated with a shape—in fact, it may be shared amongst many shapes. Modifications you make to the style's properties will apply to all shapes that share the style.

Geometric Styles

However, if you created the style object using the `GXNewStyle` function, you must then associate the style with a shape for the style modifications to have any effect. You can associate a style with a shape using the `GXSetShapeStyle` function, as shown in Listing 3-1. The `GXSetShapeStyle` function is described in *Inside Macintosh: QuickDraw GX Objects*.

Listing 3-1 Adding style information by directly manipulating a style object

```
void MakeThickPenStyle(void)
{
    gxShape aRectangleShape;
    gxStyle aThickPenStyle;

    static gxRectangle rectangleGeometry = {ff(50), ff(50),
                                           ff(200), ff(200)};

    aRectangleShape = GXNewRectangle(&rectangleGeometry);
    GXSetShapeFill(aRectangleShape, gxClosedFrameFill);

    aThickPenStyle = GXNewStyle();
    GXSetStylePen(aThickPenStyle, ff(30));

    GXSetShapeStyle(aRectangleShape, aThickPenStyle);
    GXDisposeStyle(aThickPenStyle);

    GXDrawShape(aRectangleShape);

    GXDisposeShape(aRectangleShape);
}
```

The `MakeThickPenStyle` sample function creates a rectangle shape and sets its shape fill to the closed-frame shape fill, making it a framed rectangle. The sample function then creates a new style object using the `GXNewStyle` function, which creates a style object with properties set to the standard initialized values. The owner count of this style object is 1, corresponding to the reference contained in the `aThickPenStyle` variable. The sample function then alters the pen width of the new style using the `GXSetStylePen` function.

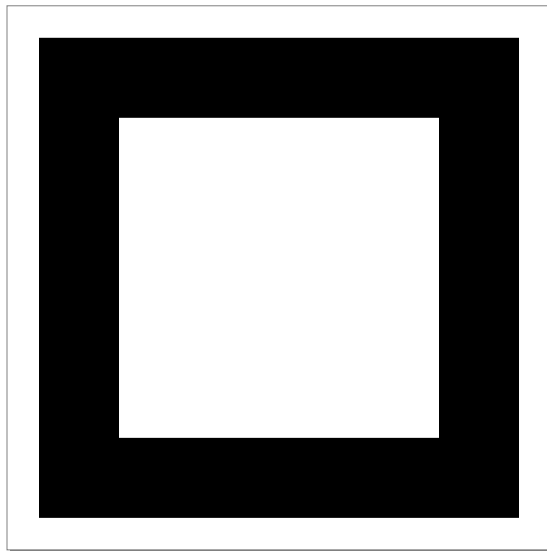
Geometric Styles

To associate the style with the rectangle shape, the sample function calls the `GXSetShapeStyle` function. This function disposes of the style previously referenced by the rectangle shape, stores a reference to the new style in the rectangle shape object, and increments the style's owner count—there are now two references to the style: one in the sample function's local variable, and one in the rectangle shape.

Finally, the sample function disposes of the style, which indicates that the reference to the style stored in the local variable `aThickPenStyle` is no longer needed. QuickDraw GX decrements the owner count of the style, which becomes 1, corresponding to the reference contained in the rectangle shape.

Finally, the sample function draws the rectangle, which appears as in Figure 3-34.

Figure 3-34 Rectangle with thick pen



The second method of altering styles involves functions that operate on style objects indirectly through the shape objects that reference them.

When using this category of function, you need only provide a reference to the shape whose style information you want to change. QuickDraw GX finds the associated style object and alters the appropriate style property for you.

Geometric Styles

In fact, QuickDraw GX provides one further level of service with this category of functions. If the shape that you specify is sharing its style with other shapes, QuickDraw GX first makes a copy of the style object, associates the copy with the shape you specified, and then alters the appropriate property of the copy.

Listing 3-2 shows an alternate approach to creating the thick-framed rectangle from Listing 3-1.

Listing 3-2 Manipulating style information indirectly

```
void MakeThickRectangle(void)
{
    gxShape aRectangleShape;

    gxRectangle rectangleGeometry = {ff(50), ff(50),
                                     ff(200), ff(200)};

    aRectangleShape = GXNewRectangle(&rectangleGeometry);
    GXSetShapeFill(aRectangleShape, gxHollowFill);

    GXSetShapePen(aRectangleShape, ff(30));

    GXDrawShape(aRectangleShape);

    GXDisposeShape(aRectangleShape);
}
```

As in Listing 3-1, this sample function creates a new framed rectangle shape. However, instead of creating a style object, altering the pen width property of the style object with the `GXSetStylePen` function, and associating the style with the rectangle shape, this sample function uses the `GXSetShapePen` function to accomplish those tasks in one step.

Since the rectangle shape is a new shape, it shares its style object with other shapes—the default rectangle shape at the very least. The `GXSetShapePen` function notices that the rectangle shape’s style is shared, so it makes a copy of this style, associates the copy with the rectangle shape, and alters the pen width property of this copy.

Geometric Styles

The result of this sample function looks exactly the same as the result of the previous sample function, shown in Figure 3-34.

For simplicity, the rest of the sample functions in this chapter use the second method for altering style properties.

Constraining Shape Geometries to Grids

The source-grid style attribute (`gxSourceGridStyle`) allows you to specify that QuickDraw GX should constrain the coordinates of a shape's geometry to integer positions before applying the shape's style and transform. Setting this style attribute does not actually change the information stored in the shape's geometry—instead, QuickDraw GX reinterprets the shape's geometry when drawing the shape.

If a shape has no style or transform modifications, setting this style attribute has the effect of snapping the shape to a 1/72-inch grid—an effect that is visible only on high-resolution devices. However, if the shape has style or transform modifications, setting this style attribute might have visible effects even on lower-resolution devices.

For example, you can use the source-grid style attribute in combination with a scaling transform to achieve the effect of constraining a shape to a grid much larger than 1/72 inch. The sample function in Listing 3-3 shows how to use this style attribute to constrain a shape to a half-inch grid.

Listing 3-3 Constraining a shape to a half-inch grid

```
void ConstrainShapeToGrid(void)
{
    gxMapping scaleToHalfInches;
    static long veeGeometry[] = {1, /* number of contours */
                                3, /* number of points */
                                fl(1.2), fl(1.1),
                                fl(2.9), fl(2.8),
                                fl(5.2), fl(.9)};

    gxShape aVeeShape;

    aVeeShape = GXNewPolygons((gxPolygons *) veeGeometry);
    GXSetShapeFill(aVeeShape, gxOpenFrameFill);
    GXSetShapePen(aVeeShape, fl(5.0 * (1.0/36.0)));
}
```

Geometric Styles

```

GXResetMapping(scaleToHalfInches);
GXScaleMapping(scaleToHalfInches, ff(36), ff(36),
               ff(0), ff(0));
GXSetShapeMapping(aVeeShape, scaleToHalfInches);

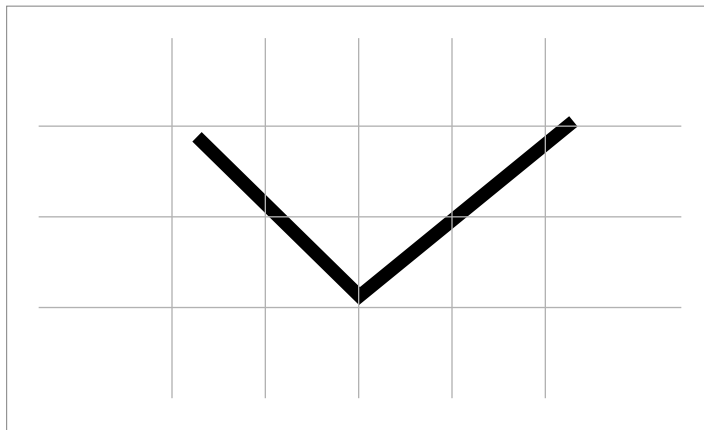
GXDrawShape(aVeeShape);

GXDisposeShape(aVeeShape);
}

```

This sample function defines a small, irregular, V-shape geometry and scales the shape up by 36 points, or half an inch. The pen width is set to 5.0 (divided by 36.0 to counteract the scaling). The result of this sample function is shown in Figure 3-35.

Figure 3-35 Scaled, but not constrained, V shape



Notice that before QuickDraw GX applies the mapping, the coordinates of the shape's geometry represent points, whereas after QuickDraw GX applies the mapping, the coordinates of the shape's geometry effectively represent half inches.

If you set the source-grid style attribute by adding this line of code to the sample function:

```

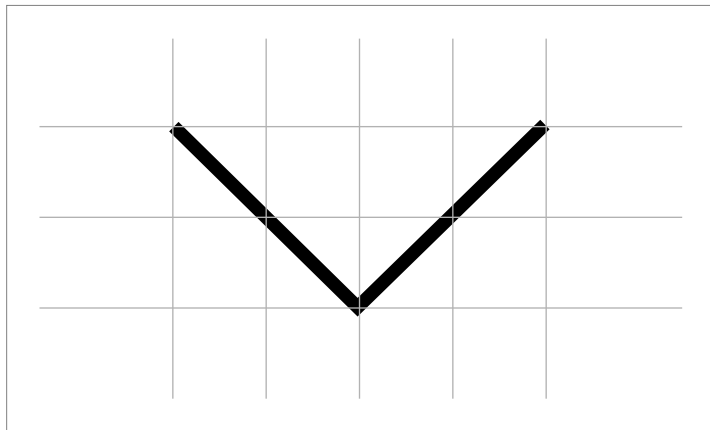
GXSetShapeStyleAttributes(aVeeShape, gxSourceGridStyle);

```

Geometric Styles

QuickDraw GX constrains the coordinates of the shape's geometry to the nearest integer position before applying the mapping. Therefore, after the mapping, the shape's geometric points lie on a half-inch grid, as shown in Figure 3-36.

Figure 3-36 Constrained V shape



The sample function in this section uses some concepts from other parts of QuickDraw GX. For more information about scaling, mappings, and transforms, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For more information about the `gxSourceGridStyle` style attribute, see “Style Attributes” on page 3-98.

Constraining Shapes to Device Grids

QuickDraw GX provides the device-grid style attribute (`gxDeviceGridStyle`), which allows you to constrain the geometric points of a shape to integer positions *after* the style, transform, and view modifications have been made.

This style attribute constrains the geometric points of a shape to the nearest integer pixel position on the device to which the shape is rendered. Unlike the source-grid style attribute, the device-grid style attribute never drastically affects the position of the shape. However, for shapes that do not have the device-grid attribute set, QuickDraw GX makes minor modifications when drawing contours whose geometric points lie between pixels; you can use the device-grid style attribute to override these modifications, which typically produces better-looking results.

Geometric Styles

The sample function in Listing 3-4 creates a star-shaped polygon and rotates it 28 degrees, which causes its geometric points to lie between integer positions.

Listing 3-4 Creating a shape with fractional geometric point positions

```
void ConstrainShapeToDeviceGrid(void)
{
    long starGeometry[] = {1, /* number of contours */
                           9, /* number of points */
                           ff(40), ff(40),
                           ff(50), ff(20),
                           ff(60), ff(40),
                           ff(80), ff(50),
                           ff(60), ff(60),
                           ff(50), ff(80),
                           ff(40), ff(60),
                           ff(20), ff(50),
                           ff(40), ff(40),
                           };

    gxShape aStar;

    aStar = GXNewPolygons((gxPolygons *) starGeometry);
    GXSetShapeFill(aStar, gxOpenFrameFill);

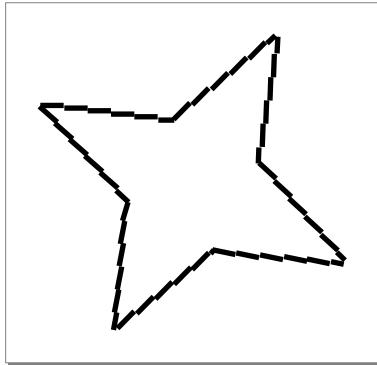
    RotateShapeAboutCenter(aStar, ff(28));
    GXDrawShape(aStar);

    GXDisposeShape(aStar);
}
```

Geometric Styles

Because the geometric points of the rotated star do not lie on integer positions, QuickDraw GX does not draw the contours of the star with the most visually appealing lines; instead, it makes minor adjustments to reflect the fractional part of the geometric point coordinates as shown in Figure 3-37.

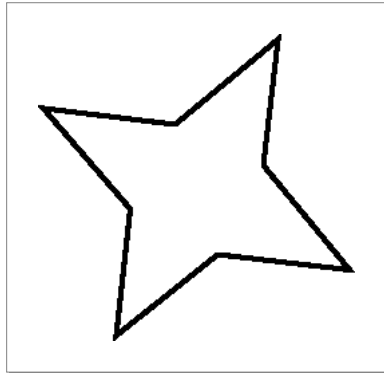
Figure 3-37 Rotated star not constrained to device grid (magnified 200 percent)



If you constrain the star shape to the device grid by adding this line of code to the sample function:

```
GXSetShapeStyleAttributes(aStar, gxDeviceGridStyle);
```

QuickDraw GX constrains the shape's geometric points to the device grid before choosing the pixels to represent the shape's contours, which creates better-looking lines, as shown in Figure 3-38.

Figure 3-38 Rotated star constrained to device grid (magnified 200 percent)

The sample function in this section uses some concepts from other parts of QuickDraw GX. For more information about rotating, mappings, and transforms, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For more information about the `gxDeviceGridStyle` style attribute, see “Style Attributes” on page 3-98.

Using Curve Error When Converting Paths to Polygons

You can use the curve error property of the style object in a variety of situations—for example, when approximating a path shape (which includes curves) with a polygon shape (which includes only straight lines).

The `GXSetShapeType` function, which is described in full in *Inside Macintosh: QuickDraw GX Objects*, allows you to convert a shape from one shape type to another. When you convert a path shape that contains curves to a polygon shape, QuickDraw GX uses the curve error of the shape’s style to determine how close to make the polygon approximation. The distance between the polygon and the original path is never greater than the number of grid points (1/72-inch units) specified by the curve error.

Geometric Styles

Listing 3-5 shows a sample function that creates a circular path shape, sets its curve error to 1, and converts it to a polygon shape.

Listing 3-5 Converting a circle to a polygon

```
void ConvertCircleToPolygon(void)
{
    gxRectangle circleBounds = {ff(50), ff(50),
                                ff(200), ff(200)};

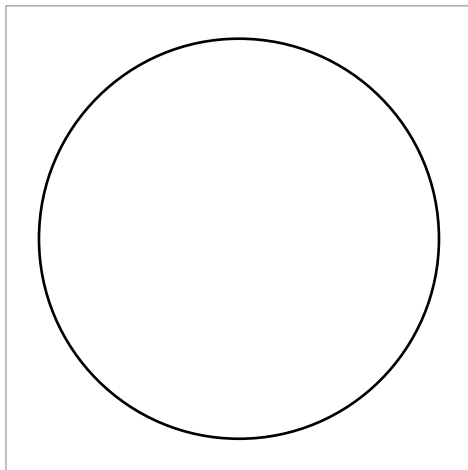
    gxShape    aCircle;

    aCircle = NewArc(&circleBounds, ff(0), ff(360), false);
    GXSetShapeFill(aCircle, gxClosedFrameFill);

    GXSetShapeCurveError(aCircle, ff(1));
    GXSetShapeType(aCircle, gxPolygonType);
    GXDrawShape(aCircle);
    GXDisposeShape(aCircle);
}
```

Since the curve error is 1 in this example, the resulting polygon is never more than 1 grid point away from the original circle, which makes for an accurate approximation, as shown in Figure 3-39.

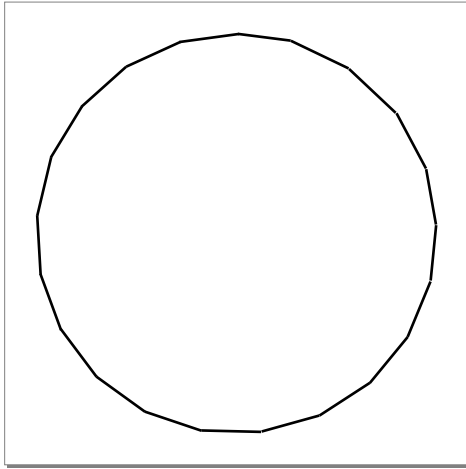
Figure 3-39 Polygon approximation of a circle with curve error of 1



Geometric Styles

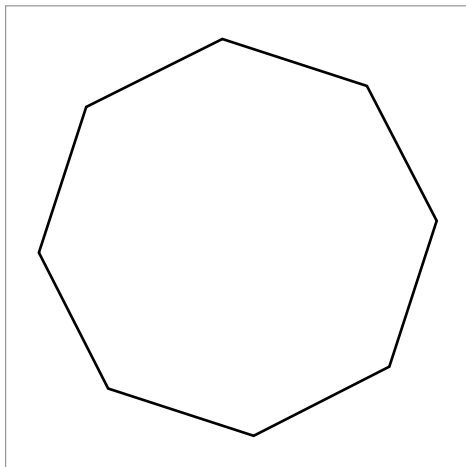
Increasing the curve error decreases the accuracy of the approximation. Setting the curve error to 5 in this example creates the polygon shown in Figure 3-40, which has fewer sides than the polygon in Figure 3-39.

Figure 3-40 Polygon approximation of a circle with curve error of 5



If you increase the curve error to 10, the octagon shown in Figure 3-41 results.

Figure 3-41 Polygon approximation of a circle with curve error of 10

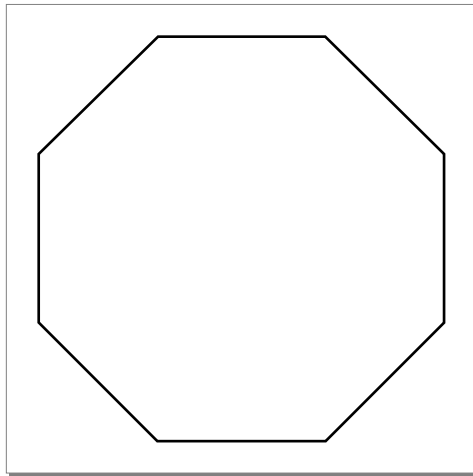


Geometric Styles

Although decreasing the curve error leads to more accurate approximations in general, a curve error of 0 is a special case. A curve error of 0 indicates that QuickDraw GX should not approximate the path at all. Instead, QuickDraw GX simply removes all off-curve control points, leaving a polygon made up of the on-curve geometric points of the initial path.

In Listing 3-5, the circular path returned by the `NewArc` library routine contains eight off-curve control points, which imply eight on-curve geometric points midway between each pair of off-curve control points. A curve error of 0 results in a polygon containing these eight on-curve points, as shown in Figure 3-42.

Figure 3-42 Polygon resulting from a curve error of 0



For more information about paths, polygons, and on-curve and off-curve geometric points, see Chapter 3, “Geometric Shapes.”

For more information about curve error and the functions you can use to manipulate it, see “Curve Error” on page 3-14 and “Getting and Setting Curve Error” on page 3-114.

Using Curve Error When Reducing Shapes

You can also use curve error to eliminate excess detail in complicated shapes. When you call the `GXReduceShape` or `GXSimplifyShape` functions, QuickDraw GX averages points within a curve error of each other.

You can use this feature to smooth a complicated contour, such as the wavy line created in Listing 3-6.

Listing 3-6 Creating a complicated contour

```
void FlattenWavyLine(void)
{
    gxShape aWave;

    static longwavyGeometry[] = {1, /* number of contours */
                                  13, /* number of points */
                                  0x2AA00000, /* 0010 0101 0101 */
                                  ff(80), ff(100), /* on */
                                  ff(110), ff(100), /* on */
                                  ff(113), ff(91), /* off */
                                  ff(118), ff(103), /* on */
                                  ff(123), ff(85), /* off */
                                  ff(128), ff(100), /* on */
                                  ff(133), ff(112), /* off */
                                  ff(135), ff(97), /* on */
                                  ff(141), ff(106), /* off */
                                  ff(145), ff(94), /* on */
                                  ff(150), ff(109), /* off */
                                  ff(153), ff(100), /* on */
                                  ff(183), ff(100) /* on */
                                };

    aWave = GXNewPaths((gxPaths *) wavyGeometry);
    GXSetShapeFill(aWave, gxOpenFrameFill);

    GXDrawShape(aWave);

    GXDisposeShape(aWave);
}
```

Geometric Styles

The shape created by this sample function is shown in Figure 3-43.

Figure 3-43 Wavy line



If you add the following lines of code to this sample function:

```
GXSetShapeCurveError(aWave, ff(10));  
GXReduceShape(aWave, 0);
```

the resulting shape has a slightly smoother appearance because QuickDraw GX averages sequential on-curve geometric points within the specified distance (a distance of 10 grid points). Figure 3-44 shows the resulting shape.

Figure 3-44 Wavy line somewhat smoothed by curve error of 10



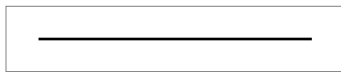
Increasing the curve error increases the number of geometric points that QuickDraw GX averages. A curve error of 15 results in the shape shown in Figure 3-45.

Figure 3-45 Wavy line smoothed by curve error of 15



A curve error of 20 results in a completely straight line—all of the points between the start point and the end point of the contour have been averaged out as shown in Figure 3-46.

Figure 3-46 Wavy line completely straightened by curve error of 20



Note

When QuickDraw GX reduces a shape, it does not ignore the first and last points of the contour. If these points had been close enough to the other points in this example, they, too, would have been averaged, and the entire shape would have become a point. ♦

For more information about curve error and the functions you can use to manipulate it, see “Curve Error” on page 3-14 and “Getting and Setting Curve Error” beginning on page 3-114.

Manipulating Pen Width and Placement

The pen width property of a style object determines the width with which QuickDraw GX draws a shape’s contours, and three of the style attributes determine where QuickDraw GX places the pen in relation to a shape’s contours. These three attributes are

- the center-frame style attribute (`gxCenterFrameStyle`)
- the inside-frame style attribute (`gxInsideFrameStyle`)
- the outside-frame style attribute (`gxOutsideFrameStyle`)

Since contour direction and crossed contours affect pen placement, the examples in this section use a path shape shaped like a figure eight, as defined in Listing 3-7.

Listing 3-7 Defining a figure eight

```
void CreateFigureEight(void)
{
    gxShape aPathShape;

    static long figureEightGeometry[] = {1, /* number of contours */
                                         4, /* number of points */
                                         0xF0000000, /* 1111 ... */
                                         ff(50), ff(50),
                                         ff(200), ff(200),
                                         ff(50), ff(200),
                                         ff(200), ff(50)};

    aPathShape = GXNewPaths((gxPaths *) figureEightGeometry);
    GXSetShapeFill(aPathShape, gxClosedFrameFill);

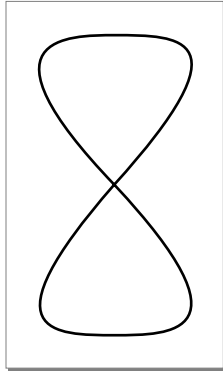
    GXDrawShape(aPathShape);

    GXDisposeShape(aPathShape);
}
```

Geometric Styles

Figure 3-47 shows the result of this sample function with the default pen width, which is a hairline, and the default pen placement, which is centered (as it always is for hairlines).

Figure 3-47 A hairline figure eight

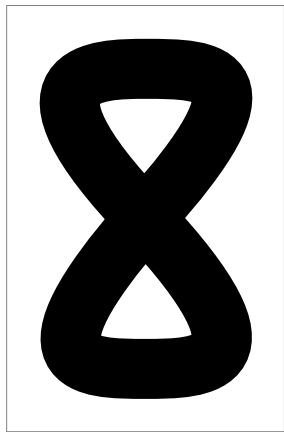


To increase the width of the pen, you can add the following line of code to the `CreateFigureEight` sample function:

```
GXSetShapePen(aPathShape, ff(30));
```

which results in the shape depicted in Figure 3-48.

Figure 3-48 A thick figure eight



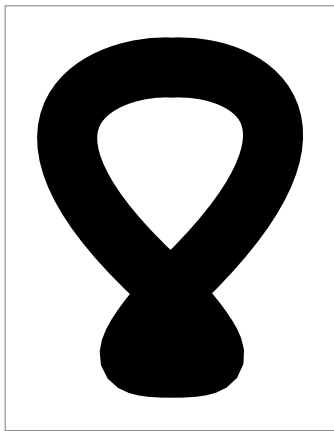
Geometric Styles

To change the placement of the thick pen, you can use the `GXSetShapeStyleAttributes` function to set the inside-frame style attribute or outside-frame style attribute. For example, if you add this line of code to the `CreateFigureEight` sample function:

```
GXSetShapeStyleAttributes(aPathShape, gxInsideFrameStyle);
```

QuickDraw GX shifts the entire pen width, which is 30 points, to the inside of the figure eight. Since, by default, QuickDraw GX defines the inside of a contour to be the right side, contour direction is significant in this case, and the resulting shape appears as depicted in Figure 3-49.

Figure 3-49 A figure eight with pen inset



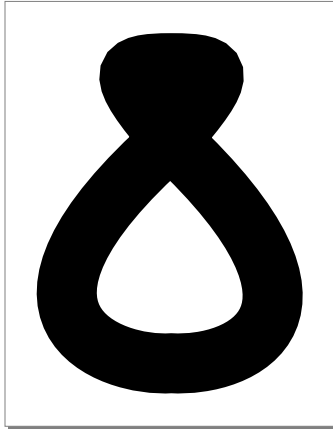
Geometric Styles

If you indicate that the pen should be placed outside—that is, to the left of—the contour, using this line of code:

```
GXSetShapeStyleAttributes(aPathShape, gxOutsideFrameStyle);
```

the figure reverses, appearing as shown in Figure 3-50.

Figure 3-50 A figure eight with pen outset



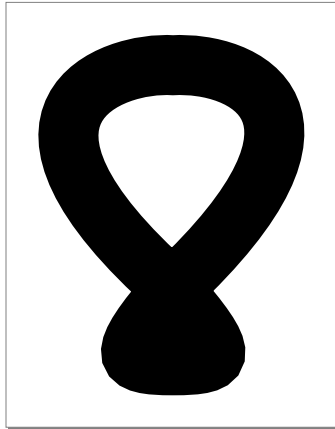
The contour direction of the path shape determines which side is the inside and which side is the outside. If you reverse the contour direction by reversing the order of the geometric points with this definition:

```
static long figureEightGeometry[] = {1, /* number of contours */
                                     4, /* number of points */
                                     0xF0000000, /* 1111 ... */
                                     ff(200), ff(50)
                                     ff(50), ff(200),
                                     ff(200), ff(200),
                                     ff(50), ff(50)};
```

but still set the outside-frame style attribute:

```
GXSetShapeStyleAttributes(aPathShape, gxOutsideFrameStyle);
```

then the resulting shape appears to be the same as the original figure-eight shape with the path *inset*, as shown in Figure 3-51.

Figure 3-51 A reversed figure eight with pen outset

However, this figure still doesn't *look* like a figure eight with the path outset—it looks like a figure eight with the upper half of the path outset and the lower half of the path inset. This problem arises because the path crosses itself. To fix this problem, you can call the `GXSimplifyShape` function, which redefines the geometry of the shape so that the path has no crossed contours. Listing 3-8 shows a sample function that uses the `GXSimplifyShape` to remove the unwanted contour crossing.

Listing 3-8 Removing unwanted contour crossings

```
void CreateUncrossedFigureEight(void)
{
    gxShape aPathShape;

    static long figureEightGeometry[] = {1, /* number of contours */
                                         4, /* number of points */
                                         0xF0000000, /* 1111 ... */
                                         ff(50), ff(50), /* off */
                                         ff(200), ff(200), /* off */
                                         ff(50), ff(200), /* off */
                                         ff(200), ff(50)}; /* off */

    aPathShape = GXNewPaths((gxPaths *) figureEightGeometry);
    GXSetShapeFill(aPathShape, gxClosedFrameFill);
    GXSetShapePen(aPathShape, ff(30));
    GXSimplifyShape(aPathShape);
}
```

Geometric Styles

```

GXSetShapeStyleAttributes(aPathShape, gxAutoInsetStyle);
GXSetShapeStyleAttributes(aPathShape, gxOutsideFrameStyle);

GXDrawShape(aPathShape);

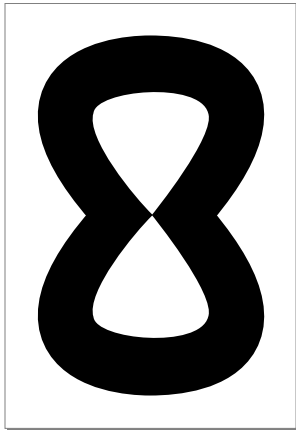
GXDisposeShape(aPathShape);
}

```

This sample function calls `GXSimplifyShape` to uncross the contours of the figure eight. However, you cannot be sure whether `GXSimplifyShape` uncrosses the contours by reversing the direction of the upper half of the figure eight, making each loop clockwise, or by reversing the lower half of the figure eight, making each loop counterclockwise. Therefore, the `CreateUncrossedFigureEight` sample function sets the auto-inset style attribute, which overrides the default assumption that the right side of the contour is the inside. Instead, QuickDraw GX determines the true inside of each contour.

Finally, now that the contours of the path do not cross and QuickDraw GX is determining the actual inside of the contour, setting the outside-frame style attribute works more as you would expect, as shown in Figure 3-52.

Figure 3-52 Uncrossed figure eight with pen outset



For more information about pen placement, see “Pen Placement” beginning on page 3-18. For more information about style attributes, see “Style Attributes” beginning on page 3-17 and “Style Attributes” beginning on page 3-98.

Adding Caps to a Shape

To add a cap shape to the ends of another shape's contours, you must create a cap record structure. The cap structure has three fields: one for the start cap shape, one for the end cap shape, and one for the cap attributes.

Listing 3-9 shows how to create a cap structure with an arrow head for the start cap, an arrow tail for the end cap, and no cap attributes.

Listing 3-9 Creating an arrow

```
void CreateArrow(void)
{
    gxShape aCurve, arrowHead, arrowTail;

    static gxCurve curveGeometry = {ff(25),  ff(125),
                                     ff(100), 0,
                                     ff(225), ff(125)};

    static long arrowHeadPolygonGeometry[] = {4, /* # of points */
                                              -ff(3), 0,
                                              0, fixed1,
                                              fixed1, 0,
                                              0, -fixed1};

    static long arrowTailPolygonGeometry[] = {5, /* # of points */
                                              -fixed1, 0,
                                              0, fixed1,
                                              ff(2), fixed1,
                                              ff(2), -fixed1,
                                              0, -fixed1};

    gxCapRecord theCapRecord;

    aCurve = GXNewCurve (&curveGeometry);

    arrowHead = NewPolygon((gxPolygon *)
                          &arrowHeadPolygonGeometry);
    arrowTail = NewPolygon((gxPolygon *)
                          &arrowTailPolygonGeometry);

    theCapRecord.startCap = arrowHead;
    theCapRecord.endCap = arrowTail;
    theCapRecord.attributes = gxNoAttributes;
}
```

Geometric Styles

```

    GXSetShapeCap(aCurve, &theCapRecord);

    GXDisposeShape(arrowHead);
    GXDisposeShape(arrowTail);

    GXSetShapePen(aCurve, ff(10));

    GXDrawShape(aCurve);

    GXDisposeShape(aCurve);
}

```

This sample function creates two polygon shapes: one for the arrow head and one for the arrow tail. It then creates a cap structure that contains references to the two shape objects and an attributes field with no attributes set.

The sample function then calls the `GXSetShapeCap` function, which sets the cap property of the curve shape's style object. (Remember, it makes a copy of this style object if the style is shared amongst multiple shapes.)

When the `GXSetShapeCap` function copies the start cap and the end cap from the cap record to the curve's style object, it does not simply copy references to the arrow head polygon and the arrow tail polygon. Instead, it makes copies of those shapes and includes the copies in the cap property of the curve's style object. After setting the curve shape's caps, you may subsequently make changes to the arrow head and arrow tail shapes without affecting the start cap or end cap of the curve.

Note

Actually, the `GXSetShapeCap` function does not copy the entire start cap shape or end cap shape. Instead, it copies only the geometric information of the start and end cap shapes. For this reason, you must provide start cap shapes and end cap shapes in their primitive forms. ♦

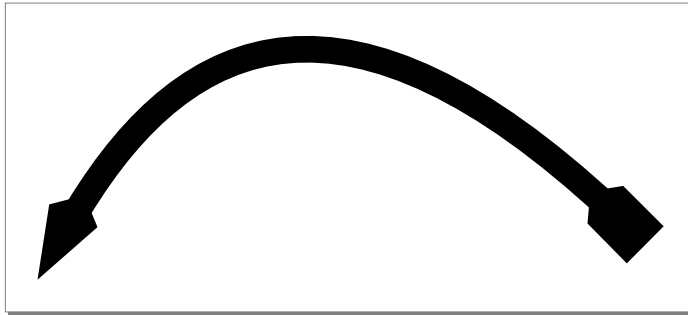
After the `CreateArrow` sample function sets the caps of the curve shape, it disposes of the arrow head and arrow tail polygons. At this point, the owner count of these shapes becomes 0 (since the curve's style does not actually reference these shapes), and the memory used by the two polygon shapes is freed.

Note

In the same way that the `GXSetShapeCap` function copies geometry information from the start and end cap shapes into a style's cap property, the `GXGetShapeCap` function creates new shape objects and copies the geometry information from a style's cap property into the new shapes. If you use the `GXGetShapeCap` function to find the caps of a shape and alter those caps, you must use the `GXSetShapeCap` function to copy your changes back into the shape's caps. ♦

Figure 3-53 shows the result of the `CreateArrow` sample function.

Figure 3-53 An arrow



Notice that QuickDraw GX rotates the start cap and the end cap to match the slope of the curve's contour, and scales them by the width of the pen. You can suppress the rotation by setting the level start-cap attribute and the level end-cap attribute.

The sections "The Cap Structure" on page 3-99 and "Cap Attributes" on page 3-101 describe the cap structure and the cap attributes in more detail, and the section "Getting and Setting Caps" beginning on page 3-123 describes the functions you can use to manipulate caps.

Adding Standard Caps to a Shape

Two types of caps that you may frequently want to add to your shapes are the round cap and the square cap. The sample function in Listing 3-10 shows how to create these types of caps.

For a round cap, you need to create a semicircle, which you can do using the library function `NewArc`. To fit the end of a contour, the bounds of this semicircle must be set as follows:

```
gxRectangle roundCapBounds = {-fl(.5), -fl(.5), fl(.5), fl(.5)};
```

and the semicircle must start at 180 degrees and span a 180 degree arc:

```
gxRoundCap = NewArc(&roundCapBounds, ff(180), ff(180), true);
```

For a square cap to fit the end of a contour, its bounds must be set as follows:

```
gxRectangle squareCapBounds = {-ff(.5), -ff(.5), ff(0), ff(.5)};
```

Geometric Styles

Listing 3-10 shows how to create a round cap and a square cap for the curve shape from previous examples.

Listing 3-10 Adding round caps and square caps to a curve

```
void CreateMyShape(void)
{
    gxShape    aCurve, gxRoundCap, gxSquareCap;

    static gxCurve curveGeometry = {ff(25),  ff(125),
                                    ff(100),  0,
                                    ff(225),  ff(125)};

    static gxRectangle roundCapBounds = {-fl(.5), -fl(.5),
                                          fl(.5),  fl(.5)};

    static gxRectangle squareCapBounds = {-ff(.5), -ff(.5),
                                           ff(0),  ff(.5)};

    gxCapRecord theCapRecord;

    aCurve = GXNewCurve (&curveGeometry);

    gxRoundCap = NewArc(&roundCapBounds, ff(180), ff(180), true);
    gxSquareCap = GXNewRectangle(&squareCapBounds);

    theCapRecord.startCap = gxRoundCap;
    theCapRecord.endCap = gxSquareCap;
    theCapRecord.attributes = gxNoAttributes;

    GXSetShapeCap(aCurve, &theCapRecord);

    GXDisposeShape(gxRoundCap);
    GXDisposeShape(gxSquareCap);

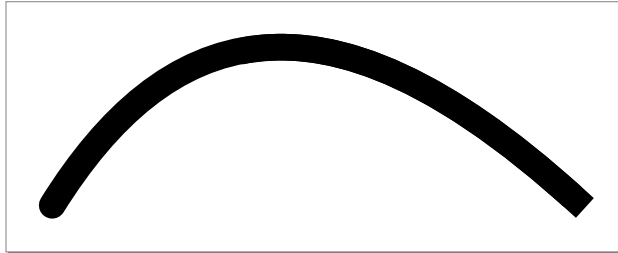
    GXSetShapePen(aCurve, ff(10));

    GXDrawShape(aCurve);

    GXDisposeShape(aCurve);
}
```


Figure 3-54 shows the result of this sample function.

Figure 3-54 Round and square caps



Notice that QuickDraw GX rotates and resizes the caps to fit the contour.

The sections “The Cap Structure” on page 3-99 and “Cap Attributes” on page 3-101 describe the cap structure and the cap attributes in more detail, and the section “Getting and Setting Caps” beginning on page 3-123 describes the functions you can use to manipulate caps.

Adding Joins to a Shape

To add a join shape to the corners of another shape’s contours, you must create a join structure. The join structure has three fields: one for the join shape, one for the join attributes, and one for the miter, which is used only for sharp joins.

Listing 3-11 shows how to create a join structure with an diamond shape as the join shape, and then apply the diamond join shape to the corners of a rectangle shape.

Listing 3-11 Adding joins to a shape

```
void CreateJoinedSquare(void)
{
    gxShape  aSquareShape, aDiamondShape;

    static gxRectangle squareGeometry = {ff(50), ff(50),
                                         ff(150), ff(150)};

    static long diamondGeometry[] = {1, /* number of contours */
                                     4, /* number of points */
                                     ff(0), ff(3),
                                     ff(1), fl(0),
                                     ff(0), -ff(3),
                                     -ff(1), ff(0)};
```

Geometric Styles

```

    gxJoinRecord theJoinRecord;

    aSquareShape = GXNewRectangle(&squareGeometry);
    GXSetShapeFill(aSquareShape, gxClosedFrameFill);

    aDiamondShape = GXNewPolygons((gxPolygons *) diamondGeometry);

    theJoinRecord.attributes = gxNoAttributes;
    theJoinRecord.join = aDiamondShape;
    theJoinRecord.miter = 0;

    GXSetShapeJoin(aSquareShape, &theJoinRecord);

    GXDisposeShape(aDiamondShape);

    GXSetShapePen(aSquareShape, ff(10));

    GXDrawShape(aSquareShape);

    GXDisposeShape(aSquareShape);
}

```

This sample function creates a square as the shape to add joins to and a diamond-shaped polygon to use for the joins. It then creates a join structure which contains a reference to the diamond shape, an attributes field with no attributes set, and a miter of 0.

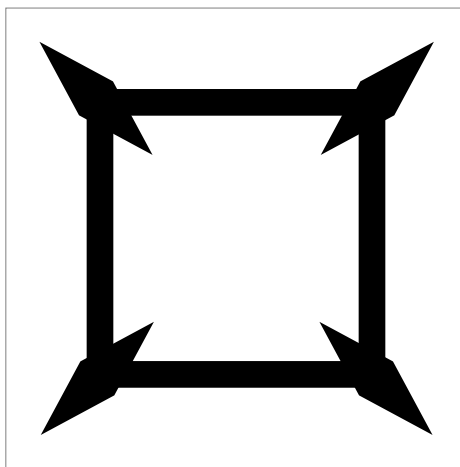
The sample function then calls the `GXSetShapeJoin` function, which sets the join property of the square shape's style object. (Remember, it makes a copy of this style object if the style is shared amongst multiple shapes.)

Note

As with caps, QuickDraw GX copies only the geometric information of the join shape into the join property of the style object; it does not copy the entire join shape. For this reason, join shapes must be in their primitive form. Once you have called `GXSetShapeJoin`, you are free to change the original join shape without affecting the joins that you have already added to a shape. ♦

After the `CreateJoinedSquare` sample function sets the joins of the square shape, it disposes of the diamond-shaped polygon. At this point, the owner count of this polygon shape becomes 0 and the memory used by the polygon shape is freed.

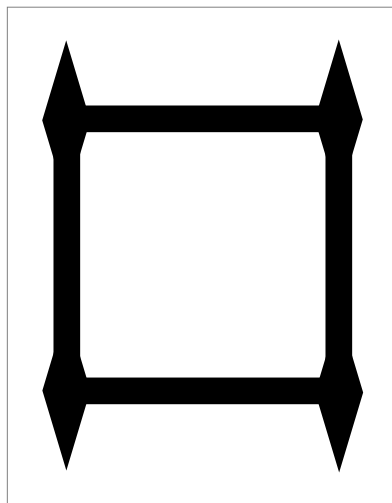
Figure 3-55 shows the result of the `CreateJoinedSquare` sample function.

Figure 3-55 A square with diamond-shaped joins

Notice that QuickDraw GX scales the join shape by the pen width and rotates the join shape to match the mid-angle of the two line segments that make each corner. You can suppress the rotation by setting the level join attribute:

```
theJoinRecord.attributes = gxLevelJoin;
```

Figure 3-56 shows the result of setting this attribute.

Figure 3-56 A square with level joins

Geometric Styles

The sections “The Join Structure” on page 3-101 and “Join Attributes” on page 3-102 describe the join structure and join attributes in more detail, and the section “Getting and Setting Joins” beginning on page 3-129 describes the functions you can use to manipulate joins.

The next section shows how to create standard joins and how to use the miter field of the join structure.

Adding Standard Joins to a Shape

Two types of joins that you may frequently want to add to your shapes are the round join and the square join. Unlike the standard cap shapes, which you add yourself by creating a semicircle shape or a half-square shape, the standard join shapes are provided for you by QuickDraw GX.

To create a standard join shape, you set the `join` field of the join record to `nil`, which indicates that you are not providing a join shape, and you set the sharp join attribute or the curve join attribute, which indicates that you want QuickDraw GX to generate one of the standard joins for you.

Listing 3-12 shows how to add a sharp join to an angle shape.

Listing 3-12 Adding a sharp join to an angle shape

```
void CreateSharpJoin(void)
{
    gxShape    anAngleShape;

    static long angleGeometry[] = {1, /* number of contours */
                                   3, /* number of points */
                                   ff(20), ff(20),
                                   ff(250), ff(60),
                                   ff(20), ff(100)};

    gxJoinRecord theJoinRecord;

    anAngleShape = GXNewPolygons((gxPolygons *) angleGeometry);
    GXSetShapeFill(anAngleShape, gxOpenFrameFill);

    theJoinRecord.attributes = gxSharpJoin;
    theJoinRecord.join = nil;
    theJoinRecord.miter = gxPositiveInfinity;

    GXSetShapeJoin(anAngleShape, &theJoinRecord);

    GXSetShapePen(anAngleShape, ff(15));
}
```

Geometric Styles

```

GXDrawShape(anAngleShape);

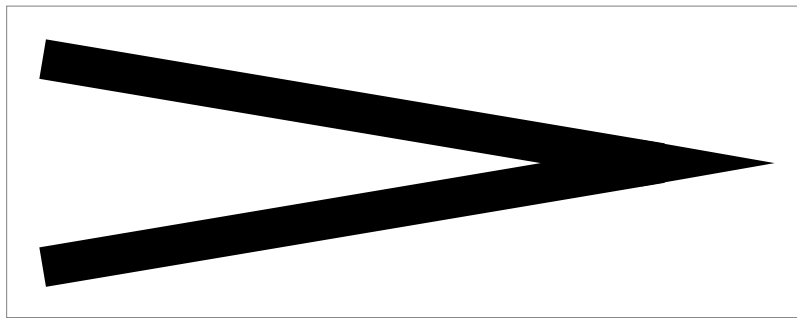
GXDisposeShape(anAngleShape);
}

```

Notice that this sample function sets the miter field to the constant value `gxPositiveInfinity`, which indicates the join should be as sharp as necessary.

Figure 3-57 shows the result of this sample function.

Figure 3-57 An angle with a sharp join

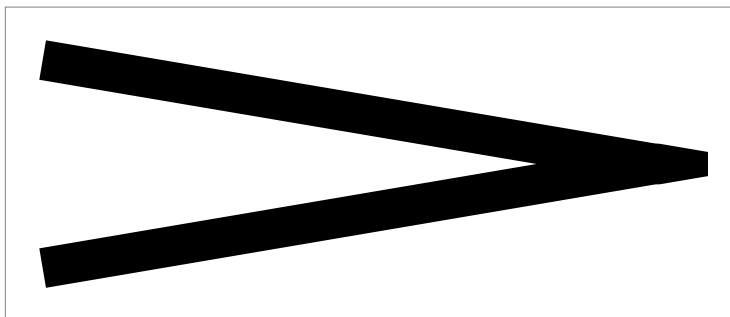


If you limit the miter of the sharp join, for example, with the code

```
theJoinRecord.miter = ff(1); /* scaled by pen width */
```

QuickDraw GX limits the distance between the actual corner of the contour as specified in the shape's geometry and the tip of the corner as actually drawn. Since miter is scaled by pen width, and the pen width in this example is 15, QuickDraw GX truncates the sharp join 15 points away from the actual corner of the geometry, as shown in Figure 3-58.

Figure 3-58 An angle with a truncated sharp join



Geometric Styles

The sections “The Join Structure” on page 3-101 and “Join Attributes” on page 3-102 describe the join record structure and the join attributes in more detail, and the section “Getting and Setting Joins” beginning on page 3-129 describes the functions you can use to manipulate joins.

Dashing a Shape

To add a dash shape along the contours of another shape, you must create a dash structure. The dash structure has five fields:

- the dash attributes, which modify the way the shape is dashed
- the dash shape, which contains the shape to use as the dashes
- the dash advance, which determines the number of points between the start of one dash and the start of the next
- the dash phase, which determines how far into the advance the dashing should start
- the dash scale, which you can use to counteract the automatic scaling of the dash shape

The sample function in Listing 3-13 creates a curve shape dashed with diamonds. First, it creates the curve shape and the diamond shape. The diamond shape has a height and a width of 30.0 points.

The sample function then creates a dash structure for the diamond dashes, and calls the `GXSetShapeDash` function to set the dash property of the curve shape’s style object.

Listing 3-13 Creating a curve shape dashed with diamonds

```
void CreateDashedCurve(void)
{
    gxShape  aCurveShape, aDiamondShape;

    static gxCurve curveGeometry = {ff(50), ff(125),
                                    ff(125), 0,
                                    ff(250), ff(125)};

    static long diamondGeometry[] = {1, /* number of contours */
                                     4, /* number of points */
                                     ff(0), ff(15),
                                     ff(15), fl(0),
                                     ff(0), -ff(15),
                                     -ff(15), ff(0)};

    gxDashRecord theDashRecord;
```

Geometric Styles

```

aCurveShape = GXNewCurve (&curveGeometry);

aDiamondShape = GXNewPolygons((gxPolygons *) diamondGeometry);

theDashRecord.attributes = gxNoAttributes;
theDashRecord.dash = aDiamondShape;
theDashRecord.advance = ff(40);
theDashRecord.phase = 0;
theDashRecord.scale = ff(30);

GXSetShapeDash(aCurveShape, &theDashRecord);

GXDisposeShape(aDiamondShape);

GXSetShapePen(aCurveShape, ff(30));

GXDrawShape(aCurveShape);

GXDisposeShape(aCurveShape);
}

```

Note

As with caps and joins, QuickDraw GX copies only the geometric information of the dash shape into the dash property of the style object; it does not copy the entire dash shape. For this reason, the dash shape must be in its primitive form. Once you have called `GXSetShapeDash`, you are free to change the original dash shape without affecting the dashes of the dashed shape. ♦

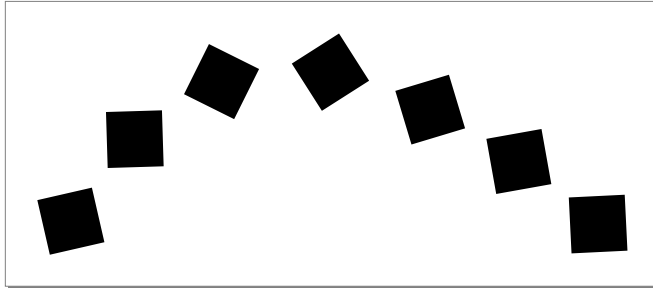
Notice that this sample function sets the dash advance to 40. Since the diamond shape is 30 points wide, this dash advance allows for 10 points between dashes. The dash phase is set to 0, which indicates that the origin of the first dash should be aligned with the beginning of the curve contour exactly.

Since QuickDraw GX scales dashes (perpendicularly to the dashed contour) by the pen width, the dashes in this example would be 900 points from tip to tip, as the diamond shape itself is 30 points high and the pen width of the curve is also 30 points. However, the sample function sets the dash scale to 30, by which QuickDraw GX scales the dashes down (again, perpendicularly to the dashed contour), which leaves the diamond shapes with their original size.

Geometric Styles

Figure 3-59 shows the result of the `CreateDashedCurve` sample function.

Figure 3-59 A dashed curve

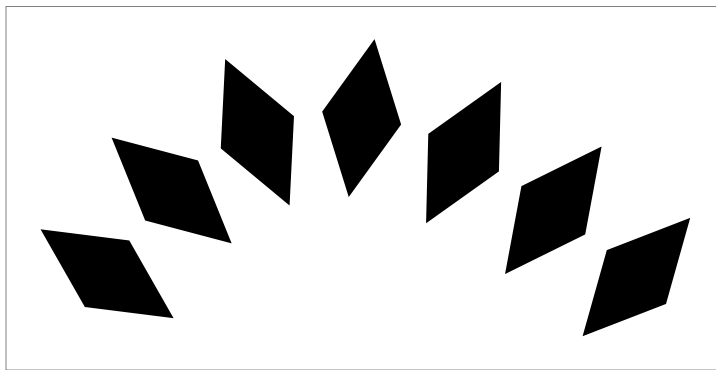


If you provide a smaller value for the dash scale, QuickDraw GX scales the dashes up in the direction perpendicular to the dashed contour. For example, if you provide a dash scale half as large:

```
theDashRecord.scale = ff(15);
```

the dashes become twice the size in the direction perpendicular to the curve, as shown in Figure 3-60.

Figure 3-60 A curve with scaled dashes



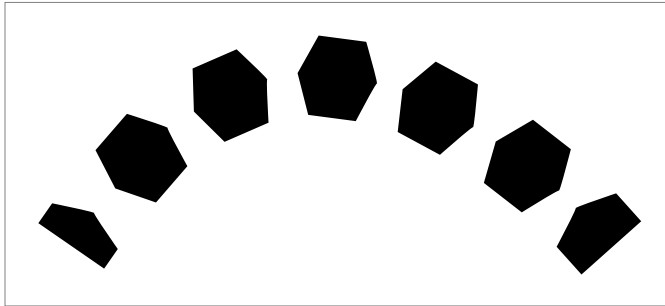
Geometric Styles

The dashes are now actually wider than the pen width of the curve. You can set the clip dash attribute to draw only the parts of the dashes that lie within the curve's pen width. For example, adding this line of code to the sample function:

```
theDashRecord.attributes = gxClipDash;
```

creates the shape shown in Figure 3-61.

Figure 3-61 A curve with clipped dashes

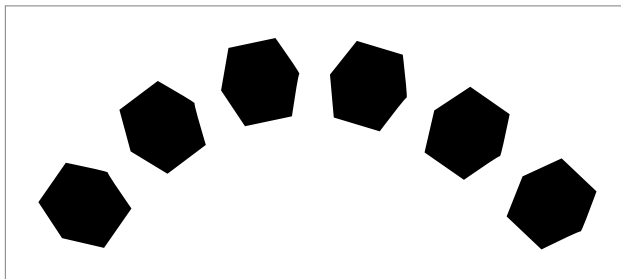


Notice that QuickDraw GX not only clips the dashes to the width of the curve, but also clips them at the ends of the curve. To shift the dashes along the curve so that you see the whole first dash, you can adjust the dash phase. For example, this line of code:

```
theDashRecord.phase = GXFloatToFract(0.50);
```

shifts the dashes forward one half of the dash advance. Since the dash advance in this case is 40, the dashes are shifted forward 20 points, as shown in Figure 3-62.

Figure 3-62 A curve with phased dashes



Geometric Styles

In this case, adjusting the dash phase is sufficient to cause a whole number of dashes to show. In other cases, you may have to use the auto-advance dash attribute, which is described in the next section.

The sections “The Dash Structure” on page 3-103 and “Dash Attributes” on page 3-105 describe the dash record and dash attributes in more detail, and the section “Getting and Setting Dashes” beginning on page 3-134 describes the functions you can use to manipulate dashes.

Adjusting Dashes to Fit Contours

Sometimes the dash advance does not divide evenly into the length of a contour and the dashes don’t look quite right. QuickDraw GX provides the auto-advance dash attribute (`gxAutoAdvanceDash`) to handle this situation.

For example, the sample function in Listing 3-14 creates a circle dashed with kite-shaped diamonds. It does not use the auto-advance dash attribute.

Listing 3-14 Creating a dashed circle

```
void CreateDashedCircle(void)
{
    gxShape  aCircleShape, aDiamondShape;

    static gxRectangle circleBounds = {ff(50), ff(50),
                                       ff(180), ff(180)};

    static long diamondGeometry[] = {1, /* number of contours */
                                     4, /* number of points */
                                     ff(0), ff(20),
                                     ff(15), ff(0),
                                     ff(0), -ff(40),
                                     -ff(15), ff(0)};

    gxDashRecord theDashRecord;

    aCircleShape = NewArc(&circleBounds, ff(0), ff(360), false);
    GXSetShapeFill(aCircleShape, gxHollowFill);

    aDiamondShape = GXNewPolygons((gxPolygons *) diamondGeometry);

    theDashRecord.attributes = gxNoAttributes;
    theDashRecord.dash = aDiamondShape;
    theDashRecord.advance = ff(30);
}
```

Geometric Styles

```

    theDashRecord.phase = 0;
    theDashRecord.scale = ff(60);

    GXSetShapeDash(aCircleShape, &theDashRecord);

    GXDisposeShape(aDiamondShape);

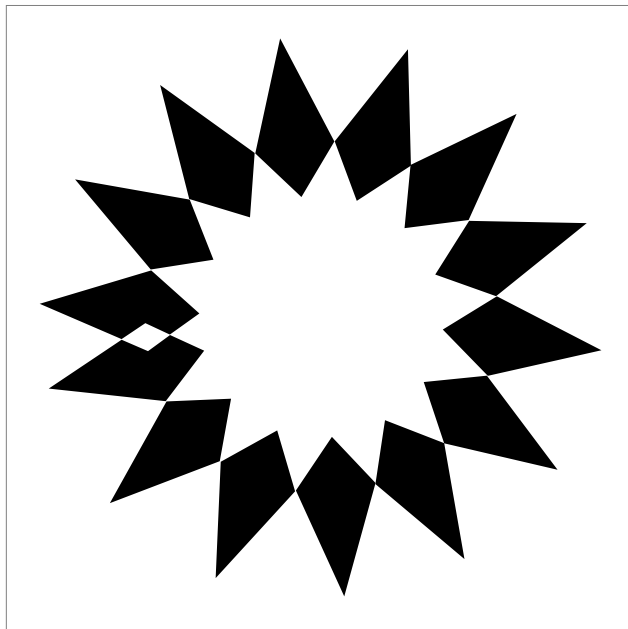
    GXSetShapePen(aCircleShape, ff(60));

    GXDrawShape(aCircleShape);
}

```

Since this sample function does not set the auto-advance dash attribute, and the dash advance of 30 does not divide evenly into the circumference of the circle, this function results in the shape shown in Figure 3-63.

Figure 3-63 Circle dashed with diamonds



Notice that the initial dash and the final dash overlap. (The overlapping region is not filled, because, by default, the dash shape has winding shape fill.)

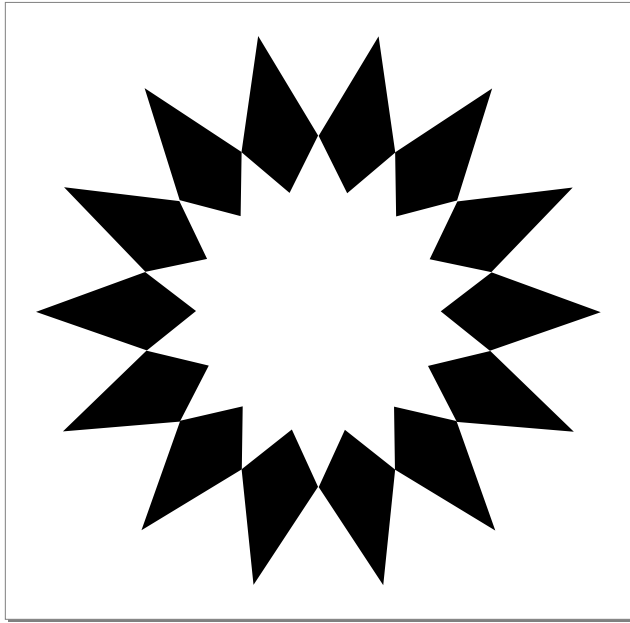
Geometric Styles

If, however, you set the auto-advance dash attribute, using this line of code:

```
theDashRecord.attributes = gxAutoAdvanceDash;
```

QuickDraw GX adjusts the dash advance accordingly. The result is shown in Figure 3-64.

Figure 3-64 Circle with automatically advanced dashes



As you can see, QuickDraw GX adjusts the dash advance the smallest amount possible to create a whole number of dashes along the contour.

The sections “The Dash Structure” on page 3-103 and “Dash Attributes” on page 3-105 describe the dash structure and dash attributes in more detail, and the section “Getting and Setting Dashes” beginning on page 3-134 describes the functions you can use to manipulate dashes.

Insetting Dashes

You can use a number of methods to change the placement of the dash shape relative to the dashed contour. For example, you can

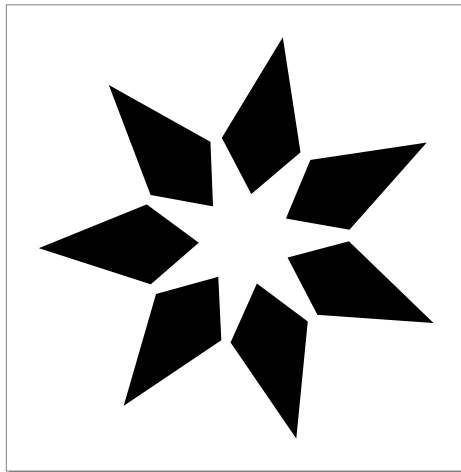
- set the inside-frame style attribute (`gxInsideFrameStyle`) or outside-frame style (`gxOutsideFrameStyle`) attribute of the style object containing the dash information so that QuickDraw GX places the dashes on the inside or outside of the contours
- change the geometry of the dash shape so that QuickDraw GX changes the placement the dash shape correspondingly when dashing the shape

These two methods produce substantially different results. For example, if you inset the pen placement in the example from the previous section by adding the call

```
GXSetShapeStyleAttributes(aCircleShape, gxInsideFrameStyle);
```

to the `CreateADashedCircle` sample function in Listing 3-14 on page 3-70, QuickDraw GX automatically adjusts the number and spacing of the dashes to fit the smaller circle, as shown in Figure 3-65.

Figure 3-65 Circle with diamond dashes inset



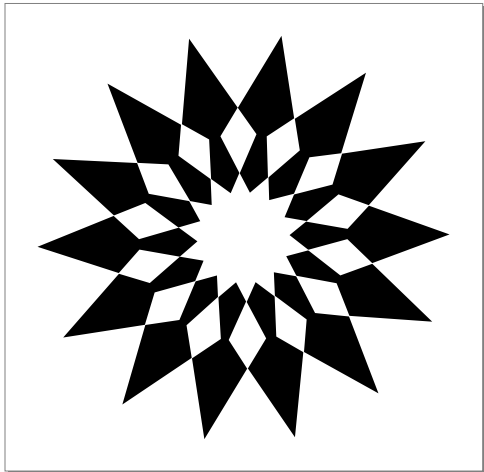
Geometric Styles

In this case, the number of dashes has been drastically reduced. If you want to keep the number of dashes constant, but move them towards the center of the circle, change the geometry of the dash shape instead of inseting the pen. For example, you can alter the diamond geometry from the `CreateDashedCircle` sample function by translating it up 30 points in the y-coordinate direction using this definition:

```
static long diamondGeometry[] = {1, /* number of contours */
                                4, /* number of points */
                                ff(0), ff(50),
                                ff(15), ff(30),
                                ff(0), -ff(10),
                                -ff(15), ff(30)};
```

In this case, if you do not inset the pen of the circle shape, the resulting shape maintains the greater number of dashes, but fits within the smaller circle, as shown in Figure 3-66.

Figure 3-66 Circle with diamond dashes moved toward the center



The sections “The Dash Structure” on page 3-103 and “Dash Attributes” on page 3-105 describe the dash structure and dash attributes in more detail, and the section “Getting and Setting Dashes” beginning on page 3-134 describes the functions you can use to manipulate dashes.

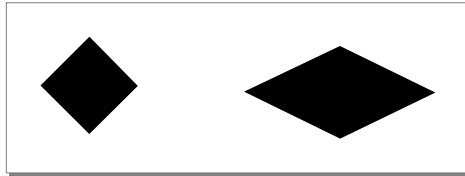
Breaking and Bending Dashes

You can use polygon shapes and path shapes as dash shapes, which means you can have a dash shape that has multiple contours. The way that QuickDraw GX places dashes along a contour can cause dashes with multiple contours to appear quite a distance from the dashed contour. QuickDraw GX provides the break dash attribute (`gxBreakDash`) and the bend dash attribute (`gxBendDash`) to address this problem.

Geometric Styles

As an example, you can create a dash shape with two entirely separate contours: for example, two separate diamonds, as shown in Figure 3-67.

Figure 3-67 Dash shape with two contours



When you use this shape to dash any sort of curve, the larger diamond falls entirely off of the contour. Listing 3-15 creates a circle shape and dashes with the double diamond shape.

Listing 3-15 Creating a dash with multiple contours

```
void CreateDoubleDiamondDash(void)
{
    gxShape    aCircleShape, aDiamondShape;

    gxRectangle circleBounds = {ff(50), ff(50), ff(180), ff(180)};

    static long doubleDiamond[] = {2, /* number of contours */
                                    4, /* number of points */
                                    ff(0), ff(10),
                                    ff(10), ff(0),
                                    ff(0), -ff(10),
                                    -ff(10), ff(0),
                                    4, /* number of points */
                                    ff(40), ff(10),
                                    ff(60), ff(0),
                                    ff(40), -ff(10),
                                    ff(20), ff(0)};

    gxDashRecord theDashRecord;

    aCircleShape = NewArc(&circleBounds, ff(0), ff(360), false);
    GXSetShapeFill(aCircleShape, gxClosedFrameFill);

    aDiamondShape = GXNewPolygons((gxPolygons *) doubleDiamond);
```

Geometric Styles

```

    theDashRecord.attributes = gxAutoAdvanceDash;
    theDashRecord.dash = aDiamondShape;
    theDashRecord.advance = ff(80);
    theDashRecord.phase = GXFloatToFract(0.0);
    theDashRecord.scale = ff(60);

    GXSetShapeDash(aCircleShape, &theDashRecord);

    GXDisposeShape(aDiamondShape);

    GXSetShapePen(aCircleShape, ff(60));

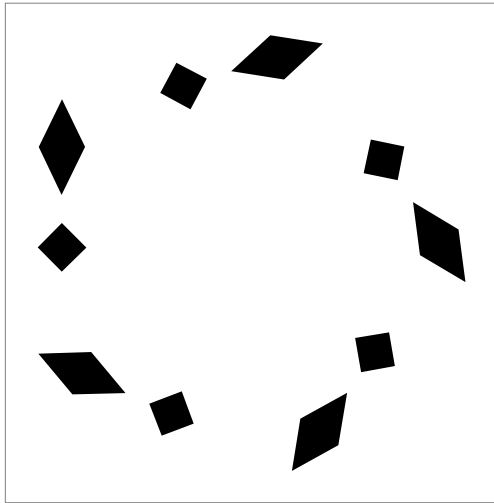
    GXDrawShape(aCircleShape);

    GXDisposeShape(aCircleShape);
}

```

This sample function creates the shape depicted in Figure 3-68.

Figure 3-68 Circle dashed with double diamonds



Geometric Styles

The break dash attribute indicates that each contour of the dash shape should be separately rotated and placed on the contours of the dashed shape. If you set the break dash attribute in this example by replacing this line of code in the sample function:

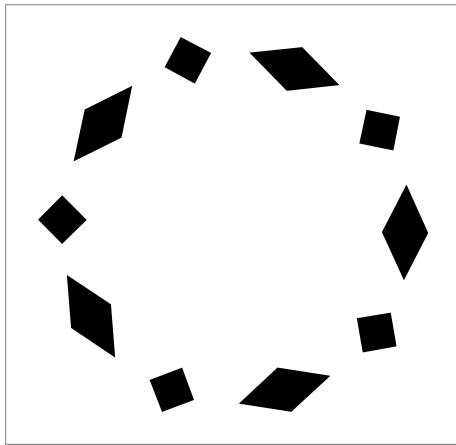
```
theDashRecord.attributes = gxAutoAdvanceDash;
```

with this line of code:

```
theDashRecord.attributes = gxAutoAdvanceDash | gxBreakDash;
```

the resulting shape appears as shown in Figure 3-69.

Figure 3-69 Circle with dashes broken



In this case, QuickDraw GX rotates and centers the large diamond contours (separately from the small diamond contours) to fit the contour of the dashed shape.

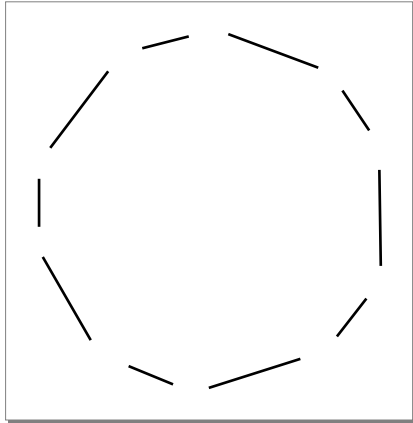
Geometric Styles

If you change the pen width of the circle in this example to 0.0, you get a hairline curve, and the dashes are mapped down to their one-dimensional image. So, for example, setting the pen width with the call

```
GXSetShapePen(aCircleShape, ff(0));
```

causes the dashed circle to appear as in Figure 3-70.

Figure 3-70 Circle with hairline dashes



QuickDraw GX provides an extra feature for hairline dashes: you can bend them to fit curved contours exactly using the bend dash attribute (`gxBendDash`).

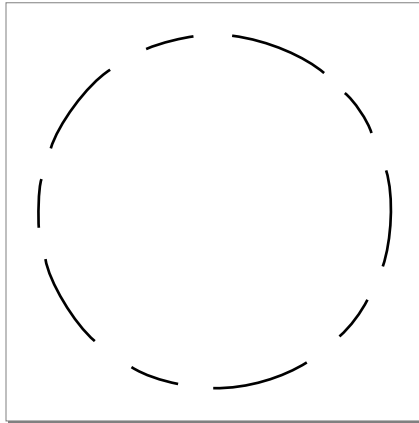
Geometric Styles

For example, if you change the dash attributes in this example using the assignment

```
theDashRecord.attributes = gxAutoAdvanceDash | gxBreakDash |  
                        gxBendDash;
```

the dashed circle appears as shown in Figure 3-71.

Figure 3-71 Circle with bent hairline dashes



Note that you can specify the bend dash attribute only for hairline contours with broken dashes.

The sections “The Dash Structure” on page 3-103 and “Dash Attributes” on page 3-105 describe the dash record structure and dash attributes in more detail, and the section “Getting and Setting Dashes” beginning on page 3-134 describes the functions you can use to manipulate dashes.

Wrapping Text to a Contour

You can wrap text to a contour by using a typographic shape as the dash shape. Since dashes must always be primitive shapes, you must convert a text or layout shape to a glyph or path shape before using it as a dash shape.

The sample function in Listing 3-16 creates a text shape, sets its font and text size, converts it to a path shape, and uses it to dash a curve.

Listing 3-16 Wrapping text

```
void WrapText(void)
{
    gxShape    aCurveShape, aTextShape;

    static gxCurve curveGeometry = {ff(25), ff(125),
                                    ff(100), 0,
                                    ff(225), ff(125)};

    gxDashRecord theDashRecord;

    aCurveShape = GXNewCurve(&curveGeometry);
    GXSetShapeFill(aCurveShape, gxOpenFrameFill);

    aTextShape = GXNewText(13,
                           (unsigned char *) "QuickDraw™ GX",
                           nil);
    SetShapeCommonFont(aTextShape, timesFont);
    GXSetShapeTextSize(aTextShape, ff(35));
    GXSetShapeType(aTextShape, gxPathType);

    theDashRecord.attributes = gxBreakDash;
    theDashRecord.dash = aTextShape;
    theDashRecord.advance = ff(330);
    theDashRecord.phase = 0;
    theDashRecord.scale = ff(35);
}
```

Geometric Styles

```

GXSetShapeDash(aCurveShape, &theDashRecord);
GXDisposeShape(aTextShape);

GXSetShapePen(aCurveShape, ff(35));

GXDrawShape(aCurveShape);

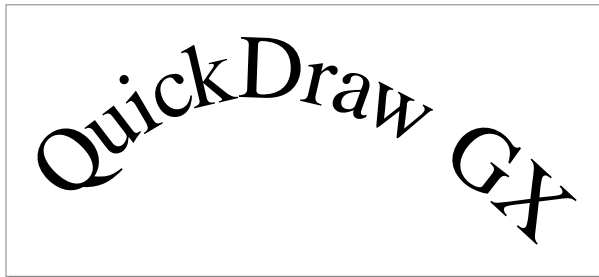
GXDisposeShape(aCurveShape);
}

```

This example sets the dash scale to equal the text size so that the glyphs do not become distorted by dash scaling.

The result of this function is depicted in Figure 3-72. Notice that QuickDraw GX rotates and places each glyph separately on the contour because the break dash attribute is set.

Figure 3-72 Wrapped text



Inside Macintosh: QuickDraw GX Typography contains more information about using typographic shapes.

Determining Dash Positions

A restriction of the QuickDraw GX dashing architecture is that each dash must be the same shape. There may be a situation where you'd like to dash a contour and have the dashes change as they progress along the contour.

To help you create the appearance of a dashed contours where the dashes change, QuickDraw GX provides the `GXGetShapeDashPositions` function. This function returns a list of mappings that identify the position and rotation of each dash on a shape.

By placing shapes in a picture using this list of mappings, you can give the effect of a contour with changing dashes.

Geometric Styles

As an example, the sample functions in this section show you how to create a picture of a clock. The `CreateDashedCircle` sample function in Listing 3-17 creates a circle with 12 dashes, each of which appears where a number would appear on a clock.

Listing 3-17 Creating a circle with 12 dashes

```
void CreateDashedCircle(void)
{
    gxShape  aCircleShape, aSquareShape;

    static gxRectangle circleBounds = {ff(50), ff(50),
                                       ff(180), ff(180)};

    static gxRectangle squareBounds = {-ff(10), -ff(10),
                                       ff(10),  ff(10)};

    gxDashRecord theDashRecord;

    aCircleShape = NewArc(&circleBounds, ff(30), ff(350), false);
    GXSetShapeFill(aCircleShape, gxClosedFrameFill);
    GXSetShapePen(aCircleShape, ff(60));

    aSquareShape = GXNewRectangle(&squareBounds);
    GXSetShapeFill(aSquareShape, gxEvenOddFill);

    theDashRecord.attributes = gxAutoAdvanceDash | gxLevelDash;
    theDashRecord.dash = aSquareShape;
    theDashRecord.advance = ff(34);
    theDashRecord.phase = 0;
    theDashRecord.scale = ff(60);

    GXSetShapeDash(aCircleShape, &theDashRecord);
    GXDisposeShape(aSquareShape);

    GXDrawShape(aCircleShape);

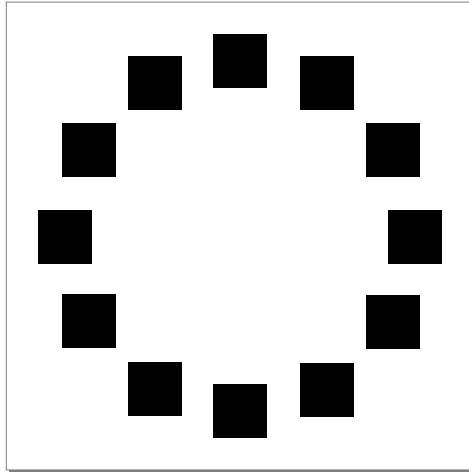
    GXDisposeShape(aCircleShape);
}
```

Geometric Styles

This sample function creates a square shape using the `GXNewRectangle` function to use as a dash for a circle shape created using the library function `NewArc`.

The result of this function is shown in Figure 3-73.

Figure 3-73 Dash positions for a clock



To replace the square dashes with numbers, the sample function in Listing 3-18 calls the `GetDashPositions` function to obtain an array of mappings that identify the position and rotation of each dash. (Notice that the dashes are not rotated in this case since the level dash attribute is set.)

The sample function in Listing 3-18 then creates a picture and adds to it text shapes containing the numbers 1 through 12. Each time text is added to the picture, its mapping is set to be the next mapping in the array of dash positions.

Listing 3-18 Creating a clock shape

```
void CreateAClock(void)
{
    gxShape  aCircleShape, aTextShape, aSquareShape, aPicture;

    static gxRectangle circleBounds = {ff(50), ff(50),
                                       ff(180), ff(180)};

    static gxRectangle squareBounds = {-ff(10), -ff(10),
                                       ff(10),  ff(10)};

    static gxPointtextPosition = {ff(0), ff(0)};
```

Geometric Styles

```

static char* numbers[] = {" 1", " 2", " 3", " 4", " 5", " 6",
                          " 7", " 8", " 9", "10", "11", "12"};

gxDashRecord theDashRecord;

long numberOfDashes, count;
gxMapping dashMappings[12];

/* Create the dashed circle from the previous example. */

aCircleShape = NewArc(&circleBounds, ff(30), ff(350), false);
GXSetShapeFill(aCircleShape, gxClosedFrameFill);

aSquareShape = GXNewRectangle(&squareBounds);
GXSetShapeFill(aSquareShape, gxEvenOddFill);

theDashRecord.attributes = gxAutoAdvanceDash | gxLevelDash;
theDashRecord.dash = aSquareShape;
theDashRecord.advance = ff(34);
theDashRecord.phase = GXFloatToFract(0.0);
theDashRecord.scale = ff(60);

GXSetShapeDash(aCircleShape, &theDashRecord);
GXSetShapePen(aCircleShape, ff(60));

/* Find the dash positions. */
numberOfDashes = GXGetShapeDashPositions(aCircleShape,
                                         dashMappings);

GXDisposeShape(aCircleShape);
GXDisposeShape(aSquareShape);

/* Create a picture with numbered text shapes. */

aTextShape = GXNewText(1, (unsigned char*) " 1",
                      &textPosition);
GXSetShapeFill(aTextShape, gxEvenOddFill);

aPicture = GXNewShape(gxPictureType);
GXSetShapeAttributes(aPicture, gxUniqueItemsShape);
for (count = 0; count <= numberOfDashes; count++) {
    GXSetShapeMapping(aTextShape, dashMappings[count]);
}

```


Geometric Styles

```

        GXSetText(aTextShape, 2, numbers[count], &textPosition);
        AddToShape(aPicture, aTextShape);
    }
    GXDisposeShape(aTextShape);

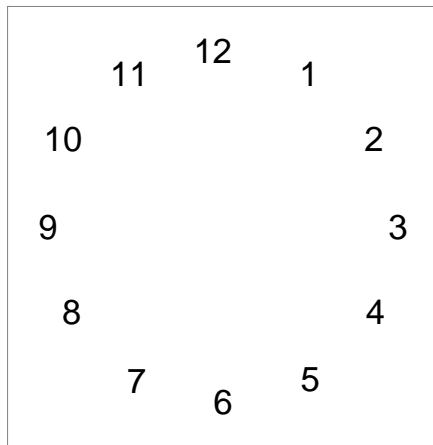
    GXDrawShape(aPicture);

    GXDisposeShape(aPicture);
}

```

The result of the `CreateAClock` sample function is depicted in Figure 3-74.

Figure 3-74 A clock shape



This sample function uses some concepts from other parts of QuickDraw GX. For more information about

- mappings, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.
- pictures and adding elements to them, see Chapter 6, “Picture Shapes.”
- typographic shapes, see *Inside Macintosh: QuickDraw GX Typography*.

Adding a Pattern to a Shape

To add a pattern to a shape, you must create a pattern structure. The pattern structure has four fields: the shape to use as the pattern, the pattern attributes, and a pair of vectors that define the grid over which QuickDraw GX places the pattern.

The sample function in Listing 3-19 creates a large rectangle shape patterned with small squares.

Listing 3-19 Patterning a shape

```
void CreatePatternedRectangle(void)
{
    gxShape    aRectangleShape, aSquarePattern;

    static gxRectangle rectangleGeometry = {ff(50), ff(50),
                                           ff(250), ff(150)};

    static gxRectangle squareGeometry = {ff(0), ff(0),
                                         ff(10), ff(10)};

    gxPatternRecord thePatternRecord;

    aRectangleShape = GXNewRectangle(&rectangleGeometry);

    aSquarePattern = GXNewRectangle(&squareGeometry);

    thePatternRecord.attributes = gxNoAttributes;
    thePatternRecord.pattern = aSquarePattern;
    thePatternRecord.u.x = ff(0);
    thePatternRecord.u.y = ff(20);
    thePatternRecord.v.x = ff(20);
    thePatternRecord.v.y = ff(0);

    GXSetShapePattern(aRectangleShape, &thePatternRecord);
    GXDisposeShape(aSquarePattern);

    GXDrawShape(aRectangleShape);

    GXDisposeShape(aRectangleShape);
}
```

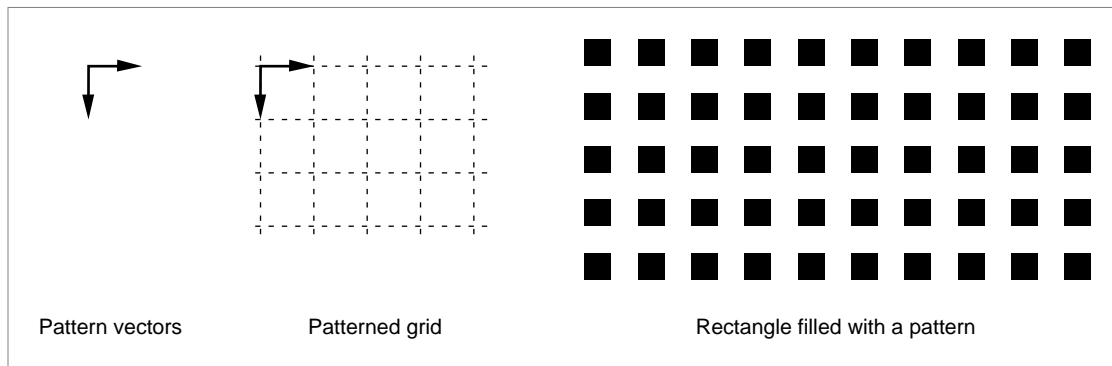
Geometric Styles

Note

As with caps, joins, and dashes, QuickDraw GX copies only the geometric information of the pattern shape into the pattern property of the style object; it does not copy the entire pattern shape. For this reason, pattern shapes must be in primitive form. Once you have called `GXSetShapePattern`, you are free to change the original pattern shape without affecting the pattern of the patterned shape. ♦

Notice that this sample function creates a square pattern shape 10 points high by 10 points wide. It places that square pattern on a rectangular grid 20 points high by 20 points wide, resulting in the shape shown in Figure 3-75.

Figure 3-75 A rectangle with a pattern



Although this example places the pattern shape on a rectangular grid, you are not limited to rectangular grids. The `u` and `v` fields of the pattern structure allow you to define a pair of vectors, so your pattern can be placed on any regular grid.

Geometric Styles

QuickDraw GX does not limit you to patterning filled shapes; you can pattern framed shapes as well. For example, if you change the previous example so that the rectangle shape is framed using the call

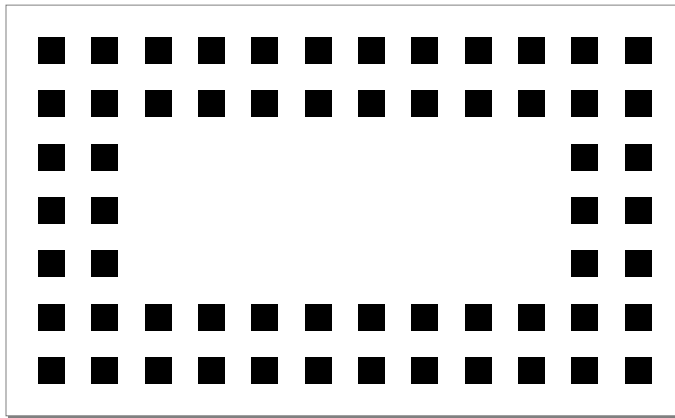
```
GXSetShapeFill(aRectangleShape, gxClosedFrameFill);
```

and has a thick pen width using the call

```
GXSetShapePen(aRectangleShape, ff(40));
```

the resulting function creates the shape shown in Figure 3-76.

Figure 3-76 A framed rectangle with a pattern



You can also pattern dashed shapes. For examples, see “Combining Caps, Joins, Dashes, and Patterns” on page 3-91.

The sections “The Pattern Structure” on page 3-106 and “Pattern Attributes” on page 3-107 describe the pattern record structure and pattern attributes in more detail, and the section “Getting and Setting Patterns” beginning on page 3-142 describes the functions you can use to manipulate patterns.

Determining Pattern Positions

As with the model for dashes, the QuickDraw GX model for patterns provides only for the case where the pattern shape remains the same throughout the entire patterned shape. If you want to pattern a shape and have the pattern change throughout it, you must use the `GXGetShapePatternPositions` function. This function returns an array of points that identify the location of each pattern shape on the patterned shape.

Geometric Styles

As an example, the sample function in this section shows you how to alter the patterned rectangle from the previous section. The sample function in Listing 3-20 first creates the patterned rectangle shown in Figure 3-75 and then uses the `GXGetShapePatternPositions` function to find the position of each small square in that patterned rectangle. The sample function then creates a picture, adding small squares at the appropriate positions, but rotating each new square a small amount.

Listing 3-20 Changing a pattern throughout a patterned shape

```
void CreateBizarrePattern(void)
{
    gxShape aRectangleShape, smallRectangle, aPicture;

    static gxRectangle rectangleGeometry = {ff(50), ff(50),
                                           ff(250), ff(150)};

    static gxRectangle smallRectGeometry = {ff(0), ff(0),
                                           ff(10), ff(10)};

    gxPatternRecord thePatternRecord;

    gxPoint *patternPositions;

    int numberOfPatterns, count;

    aRectangleShape = GXNewRectangle(&rectangleGeometry);
    GXSetShapeFill(aRectangleShape, gxEvenOddFill);

    smallRectangle = GXNewRectangle(&smallRectGeometry);
    GXSetShapeFill(smallRectangle, gxEvenOddFill);

    thePatternRecord.attributes = gxPortAlignPattern;
    thePatternRecord.pattern = smallRectangle;
    thePatternRecord.u.x = ff(0);
    thePatternRecord.u.y = ff(20);
    thePatternRecord.v.x = ff(20);
    thePatternRecord.v.y = ff(0);

    GXSetShapePattern(aRectangleShape, &thePatternRecord);

    numberOfPatterns = GXGetShapePatternPositions(aRectangleShape,
                                                  nil);
}
```

Geometric Styles

```

patternPositions = (gxPoint *)
                    NewPtr(numberOfPatterns * sizeof(gxPoint));
GXGetShapePatternPositions(aRectangleShape, patternPositions);

GXDisposeShape(aRectangleShape);

aPicture = GXNewShape(gxPictureType);
GXSetShapeAttributes(aPicture, gxUniqueItemsShape);
for (count = 0; count < numberOfPatterns; count++) {
    GXRotateShape(smallRectangle, ff(10), 0, 0);
    GXMoveShapeTo(smallRectangle, patternPositions[count].x,
                  patternPositions[count].y);
    AddToShape(aPicture, smallRectangle);
}
GXDisposeShape(smallRectangle);
DisposePtr((Ptr)patternPositions);

GXDrawShape(aPicture);

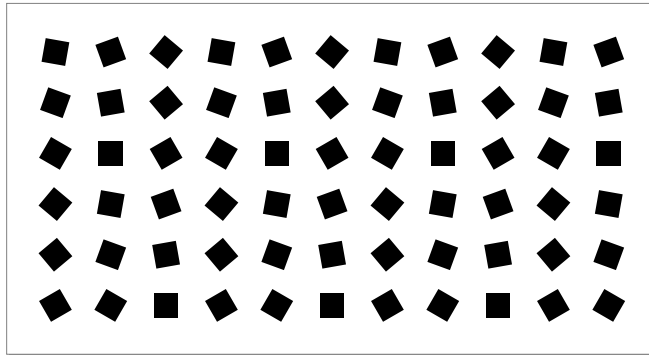
GXDisposeShape(aPicture);
}

```

This function calls the `GXGetShapePatternPositions` function twice. The first time, it sends `nil` as the value of the pattern positions array, which indicates that the `GXGetShapePatternPositions` function should not return an actual array of positions, but should return as the function result the total number of pattern positions. Once the sample function has this total, it allocates enough memory to hold the array of pattern positions, and then calls `GXGetShapePatternPositions` again to determine the actual positions.

The result of this sample function is shown in Figure 3-77.

Figure 3-77 Shape with changing pattern



Notice that, in this case, the list of positions returned by `GXGetShapePatternPositions` starts at the upper-left corner and works down each column of the patterned shape. In general, the order of the positions returned by the `GXGetShapePatternPositions` function is not guaranteed by QuickDraw GX.

This sample function uses some concepts from other parts of QuickDraw GX. For more information about

- rotating and moving shapes, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.
- pictures and adding elements to them, see Chapter 6, “Picture Shapes,” in this book.

Combining Caps, Joins, Dashes, and Patterns

As mentioned in “Interactions Between Caps, Joins, Dashes, and Patterns” on page 3-22, combining caps, joins, dashes, and patterns on the same shape causes some interesting interactions.

These elements interact differently in each of these three cases:

- the shape does not have a dash but has one or more of the three other stylistic variations
- the shape does have a dash but the clip dash attribute is not set
- the shape does have a dash and the clip dash attribute is set

Geometric Styles

When a shape has a cap and a join, QuickDraw GX adds the caps to the beginnings and ends of the shape's contours, and adds the joins to the other on-curve geometric points of the shape's contours. If the shape also has a pattern, QuickDraw GX draws this pattern throughout the shape's frame as well as the shape's caps and joins. The sample function in Listing 3-21 creates an angle shape with a round cap, a square join, and a very small square pattern.

Listing 3-21 Combining a cap, join, and pattern

```
void CapJoinPattern(void)
{
    gxShape    anAngleShape, aRoundCap, aSquareJoin, aSquarePattern;

    static long angleGeometry[] = {1, /* number of contours */
                                   3, /* number of points */
                                   ff(100), ff(100),
                                   ff(200), ff(80),
                                   ff(300), ff(100)};

    static long diamondGeometry[] = {1, /* number of contours */
                                      4, /* number of points */
                                      ff(0), ff(50),
                                      ff(10), ff(0),
                                      ff(0), -ff(50),
                                      -ff(10), ff(0)};

    static gxRectangle circleBounds = {-fl(.75), -fl(.75),
                                       fl(.75), fl(.75)};
    static gxRectangle smallSquareGeometry = {ff(0), ff(0),
                                              ff(1), ff(1)};

    gxCapRecord theCapRecord;
    gxJoinRecord theJoinRecord;
    gxPatternRecord thePatternRecord;

    /* Create the shape to be capped, joined, and patterned. */
    anAngleShape = GXNewPolygons((gxPolygons *) angleGeometry);
    GXSetShapeFill(anAngleShape, gxOpenFrameFill);
    GXSetShapePen(anAngleShape, ff(50));

    /* Create the round cap and add to the shape. */
    aRoundCap = NewArc(&circleBounds, ff(0), ff(360), false);
    theCapRecord.startCap = aRoundCap;
    theCapRecord.endCap = aRoundCap;
```


Geometric Styles

```

theCapRecord.attributes = gxNoAttributes;
GXSetShapeCap(anAngleShape, &theCapRecord);
GXDisposeShape(aRoundCap);

/* Create the square join and add to join the shape. */
aSquareJoin = GXNewRectangle(&circleBounds);
theJoinRecord.attributes = gxNoAttributes;
theJoinRecord.join = aSquareJoin;
theJoinRecord.miter = 0;
GXSetShapeJoin(anAngleShape, &theJoinRecord);
GXDisposeShape(aSquareJoin);

/* Create the small square pattern and pattern the shape. */
aSquarePattern = GXNewRectangle(&smallSquareGeometry);
GXSetShapeFill(aSquarePattern, gxSolidFill);
thePatternRecord.attributes = gxNoAttributes;
thePatternRecord.pattern = aSquarePattern;
thePatternRecord.u.x = ff(0);
thePatternRecord.u.y = ff(2);
thePatternRecord.v.x = ff(2);
thePatternRecord.v.y = ff(0);
GXSetShapePattern(anAngleShape, &thePatternRecord);
GXDisposeShape(aSquarePattern);

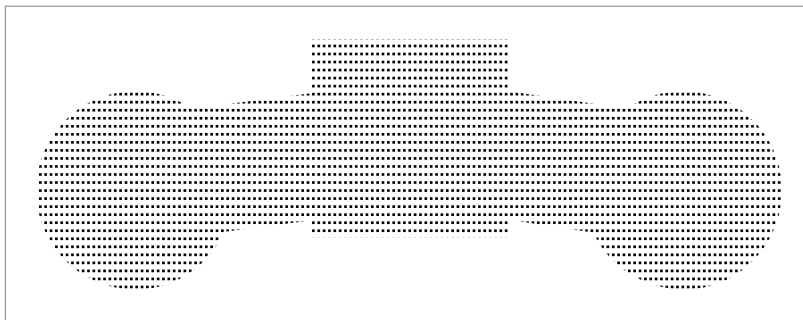
GXDrawShape(anAngleShape);

GXDisposeShape(anAngleShape);
}

```

The result of this function is shown in Figure 3-78.

Figure 3-78 Angle shape with cap, join, and pattern



Geometric Styles

The second case of cap, join, dash, and pattern interaction is when the shape has a dash but the clip dash attribute is not set. In this case, QuickDraw GX *ignores the caps and joins of the shape*. However, QuickDraw GX does draw the pattern throughout the dashes.

For example, if you add the following declarations at the appropriate places in the previous example:

```
gxShape aDiamondDash;

static long diamondGeometry[] = {1, /* number of contours */
                                4, /* number of points */
                                ff(0), ff(50),
                                ff(10), ff(0),
                                ff(0), -ff(50),
                                -ff(10), ff(0)};

gxDashRecord theDashRecord;
```

and you add the following code to create a diamond-shaped dash:

```
/* Create the diamond dash and dash the shape. */
aDiamondDash = GXNewPolygons((gxPolygons *) diamondGeometry);
GXSetShapeFill(aDiamondDash, gxEvenOddFill);
theDashRecord.attributes = gxNoAttributes;
theDashRecord.dash = aDiamondDash;
theDashRecord.advance = ff(40);
theDashRecord.phase = 0;
theDashRecord.scale = ff(50);
GXSetShapeDash(anAngleShape, &theDashRecord);
GXDisposeShape(aDiamondDash);
```

the resulting shape will appear as depicted in Figure 3-79.

Figure 3-79 Angle shape with dash and pattern; caps and join ignored



Geometric Styles

The third case of cap, join, dash, and pattern interaction is when the shape has a dash and the clip dash attribute is set. In this case, QuickDraw GX adds the cap and the join shapes to the clip shape used to clip the dashes. Patterns are not allowed in this case, so if you add the following line to the previous example:

```
theDashRecord.attributes = gxClipDash;
```

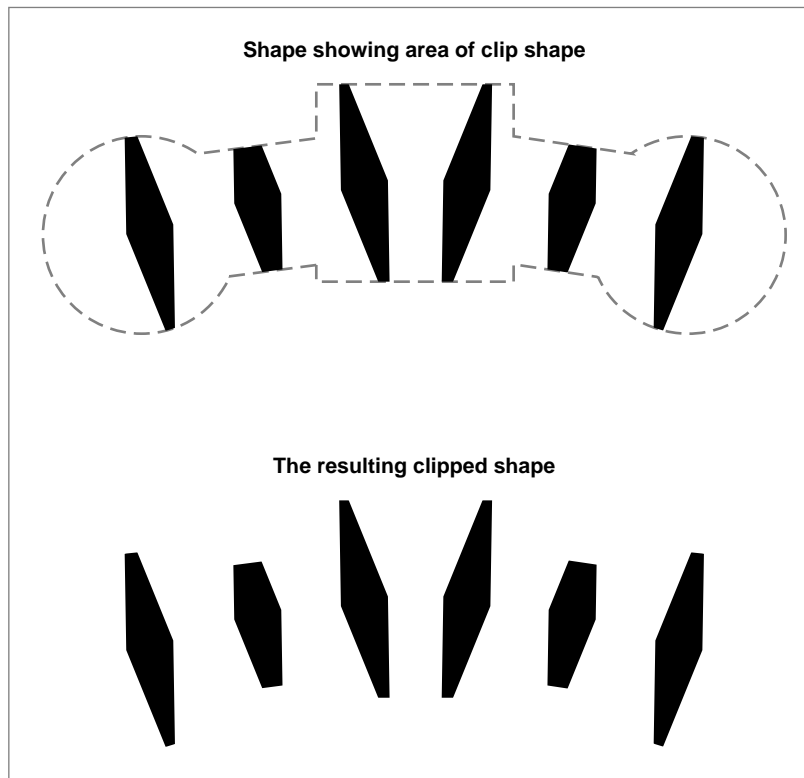
you must comment out this line:

```
/* GXSetShapePattern(anAngleShape, &thePatternRecord); */
```

which ensures that no pattern is set for the shape.

In this case, the resulting shape is drawn as shown in Figure 3-80.

Figure 3-80 Shape with cap, join, dash, and the clip dash attribute set



Notice that the dashes (which are now solid because there is no pattern) are clipped to the thick contours of the angle shape. However, at the ends and at the corner more of the dashes show because the cap shapes and the join shape are added to the clip shape used to clip the dashes.

Geometric Styles Reference

Each QuickDraw GX shape includes a shape object, a style object, an ink object, and a transform object. This section describes the data types and functions that are specific to style objects.

The “Constants and Data Types” section shows the type definition for the style object, and the structure and enumeration definitions used for five of the properties of style objects: the style attributes, the caps, the join, the dash, and the pattern.

The “Functions” section describes functions that manipulate the geometric style properties: the style attributes, the curve error, the pen width, the caps, the join, the dash, and the pattern. These properties allow you to apply stylistic variations to geometric shapes.

For information regarding creating and manipulating style objects themselves, or manipulating their tags and owner counts, see the chapter “Style Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For information regarding the typographic style properties—for example, the font, text size, and text face—see the chapter “Typographic Styles” in *Inside Macintosh: QuickDraw GX Typography*.

Constants and Data Types

This section describes the data types that you use to provide information about and to retrieve information from style objects.

You use the `gxStyle` data type when referring to a style object. This data type is described in full in the chapter “Style Objects” of *Inside Macintosh: QuickDraw GX Objects*.

You use the `gxStyleAttributes` enumeration when getting and setting individual flags of the attributes property of a style object.

You use the `gxCapRecord` structure and the `gxCapAttributes` enumeration when getting and setting the start cap and end cap of a shape.

You use the `gxJoinRecord` structure and the `gxJoinAttributes` enumeration when getting and setting the corner join of a shape.

You use the `gxDashRecord` structure and the `gxDashAttributes` enumeration when getting and setting a shape’s dashes.

You use the `gxPatternRecord` structure and the `gxPatternAttributes` enumeration when getting and setting a shape’s pattern.

Style Objects

You use the `gxStyle` data type when referring to a style object. This data type is described in full in the chapter “Style Objects” of *Inside Macintosh: QuickDraw GX Objects*.

Style objects have owner counts, tags, typographic style properties, and seven geometric style properties. The owner count and tags properties are described in *Inside Macintosh: QuickDraw GX Objects*. The typographic style properties are described in *Inside Macintosh: QuickDraw GX Typography*. The geometric style properties are listed here:

- **Style attributes.** This property is a group of flags that modify the behavior of the style object. The section “Style Attributes” on page 3-17 discusses the effects of these attributes. The section “Style Attributes” on page 3-98 describes the style attribute flags, and “Getting and Setting Style Attributes” on page 3-109 describes the functions you can use to examine or alter style attribute flags.
- **Curve error.** This property specifies the allowable amount of error when QuickDraw GX converts a path shape into a polygon shape. It also specifies how far apart geometric points must be for QuickDraw GX to consider them separate points when reducing or simplifying a shape. The section “Curve Error” on page 3-14 discusses the curve error property and the sections “Using Curve Error When Converting Paths to Polygons” on page 3-45 and “Using Curve Error When Reducing Shapes” on page 3-49 give examples of using curve error. The section “Getting and Setting Curve Error” on page 3-114 describes the functions you can use to examine or alter this property.
- **Pen width.** This property specifies the thickness of the pen QuickDraw GX uses to draw the contours of a shape. “The Geometric Pen” on page 3-15 describes how QuickDraw GX uses the pen when drawing, and “Getting and Setting the Pen Width” beginning on page 3-119 describes the functions you can use to examine or alter the pen width.
- **Cap.** This property specifies what QuickDraw GX should draw at the start and the end of a shape’s contours. The section “Caps” on page 3-23 describes start and end caps, the sections “The Cap Structure” on page 3-99 and “Cap Attributes” on page 3-101 discuss the data types you use to describe start and end caps, and the section “Getting and Setting Caps” beginning on page 3-123 describes the functions you can use to examine or alter a shape’s start and end caps.
- **Join.** This property specifies what QuickDraw GX draws at the corners of a shape’s geometry. The section “Joins” on page 3-25 describes corner joins, the sections “The Join Structure” on page 3-101 and “Join Attributes” on page 3-102 discuss the data types you use to describe corner joins, and the section “Getting and Setting Joins” beginning on page 3-129 describes the functions you can use to examine or alter a shape’s corner joins.

Geometric Styles

- **Dash.** This property specifies how QuickDraw GX should dash the contours of a shape. The section “Dashes” on page 3-27 describes dashes, the sections “The Dash Structure” on page 3-103 and “Dash Attributes” on page 3-105 discuss the data types you use to describe dashes, and the section “Getting and Setting Dashes” beginning on page 3-134 describes the functions you can use to examine or alter a shape’s dashes.
- **Pattern.** This property specifies how QuickDraw GX should fill the geometry of a shape with a pattern. The section “Patterns” on page 3-31 describes patterns, the sections “The Pattern Structure” on page 3-106 and “Pattern Attributes” on page 3-107 discuss the data types you use to describe patterns, and the section “Getting and Setting Patterns” beginning on page 3-142 describes the functions you can use to examine or alter a shape’s pattern.

Style Attributes

Each style object has a set of style attributes, which are a group of flags that modify the behavior of the style object. In particular, these flags allow you to specify how QuickDraw GX places the pen with respect to a shape’s geometry and whether the shape should be constrained to a grid. These constants are defined in the `gxStyleAttributes` enumeration:

```
enum gxStyleAttributes {
    gxCenterFrameStyle    = 0,
    gxSourceGridStyle     = 0x0001,
    gxDeviceGridStyle     = 0x0002,
    gxInsideFrameStyle    = 0x0004,
    gxOutsideFrameStyle   = 0x0008,
    gxAutoInsetStyle      = 0x0010
};
```

```
typedef long gxStyleAttribute;
```

Constant descriptions

`gxCenterFrameStyle`

Indicates that QuickDraw GX should center the geometric pen along the shape’s contours.

`gxSourceGridStyle`

Constrains the geometric points of the shape in geometry space. When drawing a shape whose style object has this flag set, QuickDraw GX moves each geometric point of the shape’s geometry to the closest integral position before applying the shape’s style and transform. (Note that the original geometric points are unchanged; this operation occurs only as the shape is being drawn.) See “Grids” beginning on page 3-20 for more information.

Geometric Styles

`gxDeviceGridStyle`

Constrains the geometric points of the shape in device space. When drawing a shape whose style object has this flag set, QuickDraw GX moves the shape's geometric points, *after* applying the shape's style and transform, to the closest integral position (that is, pixel position) in the device space. (Note that the original geometric points are unchanged; this operation occurs only as the shape is being drawn.) See “Grids” beginning on page 3-20 for more information.

`gxInsideFrameStyle`

Indicates that QuickDraw GX should position the pen along the inside of the shape's contours. By default, QuickDraw GX uses the direction of a contour to determine which side is the inside; the right side of a contour is considered the inside.

`gxOutsideFrameStyle`

Indicates that QuickDraw GX should place the pen along the outside of the shape's contours. By default, QuickDraw GX uses the direction of a contour to determine which side is the inside; the left side of a contour is considered the outside.

`gxAutoInsetStyle`

Alters the default definition of the inside and outside of a contour. When this flag is not set, QuickDraw GX assumes the right side of a contour is the inside and the left side of a contour is the outside (which provides the correct behavior for TrueType fonts). When the `gxAutoInsetStyle` flag is set, QuickDraw GX finds the true inside of each contour, regardless of the contour direction.

Setting both the `gxInsideFrameStyle` and `gxOutsideFrameStyle` style attributes results in the `inconsistent_parameters` error.

See “Grids” on page 3-20 and “Constraining Shape Geometries to Grids” beginning on page 3-40 for details about how QuickDraw GX constrains shapes to a grid. See “The Geometric Pen” on page 3-15 and “Manipulating Pen Width and Placement” on page 3-51 for examples of pen placement.

The Cap Structure

QuickDraw GX allows you to specify what to draw at the start and at the end of a shape's contours. In particular, you may specify a start cap for any point shape, and you may specify a start cap and an end cap for any line, curve, polygon, or path shape that has an open-frame shape fill.

QuickDraw GX uses the cap property of a shape's style object to store information about the start cap and end cap of the shape.

You use the cap structure when specifying cap information (using the `GXSetStyleCap` or `GXSetShapeCap` functions) and when retrieving cap information (using the `GXGetStyleCap` or `GXGetShapeCap` functions).

Geometric Styles

The cap structure is defined by the `gxCapRecord` data type:

```
struct gxCapRecord {
    gxCapAttribute attributes;
    gxShape          startCap;
    gxShape          endCap;
};
```

Field descriptions

<code>attributes</code>	Modifies the behavior of the caps. The next section, “Cap Attributes,” describes the <code>gxCapAttribute</code> flags in detail.
<code>startCap</code>	<p>Specifies what the start cap should look like. You must use shapes in their primitive form for the start cap shape. (Primitive shapes are described in detail in Chapter 4, “Geometric Operations,” in this book.) You may not use framed shapes, shapes with an inverse shape fill, full shapes, text shapes, glyph shapes, layout shapes, bitmap shapes, or picture shapes as the start cap shape.</p> <p>QuickDraw GX considers only the geometric properties (the shape type, the shape fill, and the shape geometry) of the shape specified by the <code>startCap</code> field. QuickDraw GX ignores the owner count, shape tags, and shape attributes properties and the style, ink, and transform objects of the start cap shape.</p>
<code>endCap</code>	<p>Specifies what the end cap should look like. You must use shapes in their primitive form for the end cap shape. (Primitive shapes are described in detail in Chapter 4, “Geometric Operations,” in this book.) You may not use framed shapes, shapes with an inverse shape fill, full shapes, text shapes, glyph shapes, layout shapes, bitmap shapes, or picture shapes as the end cap shape.</p> <p>QuickDraw GX considers only the geometric properties (the shape type, the shape fill, and the shape geometry) of the shape specified by the <code>endCap</code> field. QuickDraw GX ignores the owner count, shape tags, and shape attributes properties and the style, ink, and transform objects of the end cap shape.</p>

See “Caps” beginning on page 3-23, “Adding Caps to a Shape” beginning on page 3-57, and “Adding Standard Caps to a Shape” beginning on page 3-59 for examples of caps.

Cap Attributes

Each cap structure contains a set of flags that modify the way a shape is capped. These constants are defined in the `gxCapAttributes` enumeration:

```
enum gxCapAttributes {
    gxLevelStartCap= 0x0001;
    gxLevelEndCap  = 0x0002;
};

typedef long gxCapAttribute;
```

Constant descriptions

`gxLevelStartCap`

Suppresses rotation of the start cap shape. When you set this flag, QuickDraw GX does not rotate the start cap shape to match the slope of the capped contour. Instead, QuickDraw GX places the start cap shape onto the start of the capped contour with the exact orientation specified by the start cap shape's geometry.

`gxLevelEndCap`

Suppresses rotation of the end cap shape. When you set this flag, QuickDraw GX does not rotate the end cap shape to match the slope of the capped contour. Instead, QuickDraw GX places the end cap shape onto the start of the capped contour with the exact orientation specified by the end cap shape's geometry.

The Join Structure

QuickDraw GX allows you to specify a join shape to be drawn at the corners of another shape's contours. In particular, you may specify a join shape for any rectangle, polygon, or path shape that has an open-frame shape fill or a closed-frame shape fill.

- For shapes with the closed-frame shape fill, QuickDraw GX draws the specified join shape at every on-curve geometric point of each contour.
- For shapes with the open-frame shape fill, QuickDraw GX draws the specified join shape at every on-curve geometric point between the first point and the last point of each contour.

QuickDraw GX uses the join property of a shape's style object to store information about the join of the shape.

You use the join structure when specifying join information (using the `GXSetStyleJoin` or `GXSetShapeJoin` functions) and when retrieving join information (using the `GXGetStyleJoin` or `GXGetShapeJoin` functions).

Geometric Styles

The join structure is defined by the `gxJoinRecord` data type:

```
struct gxJoinRecord {
    gxJoinAttribute attributes;
    gxShape          join;
    Fixed            miter;
};
```

Field descriptions

<code>attributes</code>	Allows you to specify a level join, or to specify one of two standard types of joins: sharp joins and curve joins. The next section, “Join Attributes,” describes the <code>gxJoinAttribute</code> flags in detail.
<code>join</code>	Specifies what the join should look like. You must use shapes in their primitive form for the join shape. (Primitive shapes are described in detail in Chapter 4, “Geometric Operations,” in this book.) You may not use framed shapes, shapes with an inverse shape fill, full shapes, text shapes, glyph shapes, layout shapes, bitmap shapes, or picture shapes as the join shape. QuickDraw GX considers only the geometric properties (the shape type, the shape fill, and the shape geometry) of the shape specified by the <code>join</code> field. QuickDraw GX ignores the owner count, shape tags, and shape attributes properties and the style, ink, and transform objects of the join shape. You set this field to <code>nil</code> when you want to specify a standard join: a sharp join or curve join.
<code>miter</code>	Used to truncate sharp joins. See the next section, “Join Attributes,” for more information about sharp joins.

See “Joins” beginning on page 3-25, “Adding Joins to a Shape” beginning on page 3-61, and “Adding Standard Joins to a Shape” beginning on page 3-64 for examples of joins.

Join Attributes

Each join structure contains a set of flags that allow you to specify level joins, sharp joins, and curve joins. These constants are defined in the `gxJoinAttributes` enumeration:

```
enum gxJoinAttributes {
    gxSharpJoin= 0x0000,
    gxCurveJoin= 0x0001,
    gxLevelJoin= 0x0002
};

typedef long gxJoinAttribute;
```

Geometric Styles

Constant descriptions

<code>gxSharpJoin</code>	Indicates that QuickDraw GX should continue the outside edges of the corners of the joined shape until they meet at a point. You can use the <code>miter</code> field of the join structure to limit the size of a sharp join for very sharp corners.
<code>gxCurveJoin</code>	Indicates that QuickDraw GX should connect the outside edges of the corners of the joined shape with a circular curve.
<code>gxLevelJoin</code>	Suppresses rotation of the shape specified by the <code>join</code> field of the join structure. When you set this flag, QuickDraw GX does not rotate the join shape to match the mid-angle of the joined corner. Instead, QuickDraw GX places the join shape onto the joined corner with the exact orientation specified by the geometry of the join shape.

QuickDraw GX draws a sharp join or a curve join for every corner of every geometric shape; you may additionally specify a join shape to be added to a shape's corner using the `join` field of the join structure.

The `miter` field of the join structure allows you to limit the size of sharp joins, which is particularly useful if the joined corner is very sharp. In the `miter` field, you specify the maximum distance between the actual corner (as specified by the joined shape's geometry) and the tip of the sharp corner as drawn.

See "Adding Standard Joins to a Shape" beginning on page 3-64 for an example of a standard join.

The Dash Structure

With QuickDraw GX, you can specify that certain shapes should be drawn with dashed, instead of solid, contours. In particular, you may specify a dash for any line, curve, rectangle, polygon, or path shape that has an open-frame shape fill or a closed-frame shape fill.

QuickDraw GX uses the dash property of a shape's style object to store information about how to dash the shape.

You use the dash structure when specifying dash information (using the `GXSetStyleDash` or `GXSetShapeDash` functions) and when retrieving dash information (using the `GXGetStyleDash` or `GXGetShapeDash` functions).

The dash structure is defined by the `gxDashRecord` data type:

```
struct gxDashRecord {
    gxDashAttribute    attributes;
    gxShape             dash;
    Fixed               advance;
    fract               phase;
    Fixed               scale;
};
```

Geometric Styles

Field descriptions

attributes	Modifies the behavior of the dashes. The next section, “Dash Attributes,” describes the <code>gxDashAttribute</code> flags in detail.
dash	Specifies what the dash should look like. You must use shapes in their primitive form for the dash shape. (Primitive shapes are described in detail in Chapter 4, “Geometric Operations,” in this book.) You may not use text shapes, layout shapes, bitmap shapes, or picture shapes as the dash shape. However, you may use framed shapes and glyph, and you may also use shapes with an inverse shape fill if the clip dash attribute is set. QuickDraw GX considers only the geometric properties (the shape type, the shape fill, and the shape geometry) of the shape specified by the <code>dash</code> field. QuickDraw GX ignores the owner count, shape tags, and shape attributes properties and the style, ink, and transform objects of the dash shape.
advance	Indicates the distance between dashes. This fixed-point value is the distance along the contours of the dashed shape between the beginning of a dash and the beginning of the following dash. The value must be greater than 0.
phase	Specifies the initial placement of a dash. This value can vary between -2.0 and 2.0 . A value of 0 indicates that the dash shape should not be offset—that is, the start of the first dash shape should be aligned with the start of the contour. A value greater than 0 indicates that the first dash along the contour should begin a certain percentage into the dash shape. A value of 1.0 indicates that the dashes should be shifted exactly one advance width—this value is equivalent to specifying a value of 0. Values greater than 1.0 are equivalent to their fractional part.
scale	Specifies the scaling of the dash shape. QuickDraw GX scales the dash shape in one dimension—perpendicularly to the contour being dashed. The factor it uses to scales the dash shape in this dimension is the pen width divided by the dash scale. Therefore, decreasing the dash scale has the effect of thickening the dashed contour.

See “Dashes” beginning on page 3-27 for more information about dashes, and see page 3-66 through page 3-81 for examples of dashing.

Dash Attributes

Each dash structure contains a set of flags that modify the way a shape is dashed. These constants are defined in the `gxDashAttributes` enumeration:

```
enum gxDashAttributes {
    gxBendDash      = 0x0001;
    gxBreakDash     = 0x0002;
    gxClipDash      = 0x0004;
    gxLevelDash     = 0x0008;
    gxAutoAdvanceDash = 0x0010;
};

typedef long gxDashAttribute;
```

Constant descriptions

<code>gxBendDash</code>	Distorts the dash shape to match the contour being dashed. A dash may have the <code>gxBendDash</code> attribute only when the dashed shape's pen width is zero, indicating hairline contours. (Any other pen width results in an error condition.) When the <code>gxBendDash</code> attribute is set, QuickDraw GX maps the dash shape onto the x-axis (so that it becomes one-dimensional) and bends this flattened dash shape along the contours of the shape being dashed.
<code>gxBreakDash</code>	Indicates that QuickDraw GX should rotate and place each contour of the dash shape separately. When this attribute is set, QuickDraw GX calculates the center point of each contour of the dash shape and rotates and centers it appropriately along the contour of the shape being dashed. See Figure 3-25 on page 3-30 for an example.
<code>gxClipDash</code>	Indicates that QuickDraw GX should clip the dashes to the pen width of the dashed shape. See Figure 3-24 on page 3-29 for an example. This attribute causes dashes to have some complicated interactions with caps and joins. See the section "Interactions Between Caps, Joins, Dashes, and Patterns" on page 3-33 and "Combining Caps, Joins, Dashes, and Patterns" beginning on page 3-91 for more information.
<code>gxLevelDash</code>	Suppresses rotation of the dash shape. When this attribute is set, QuickDraw GX does not rotate the dash shape to match the slope of the dashed shape's contours. Instead, QuickDraw GX places the dash shape onto the contours of the dashed shape with the exact orientation specified by the geometry of the dash shape.
<code>gxAutoAdvanceDash</code>	Adjusts the dash advance so that a whole multiple of dash shapes fit each contour.

Geometric Styles

These sections include examples of using dash attributes:

- “Dashing a Shape” on page 3-66
- “Adjusting Dashes to Fit Contours” on page 3-70
- “Breaking and Bending Dashes” on page 3-74

The Pattern Structure

With QuickDraw GX, you can specify that certain shapes be patterned. For shapes with solid shape fills, QuickDraw GX fills the shape by repeating a pattern shape that you specify, over a grid that you specify.

You can also pattern shapes with framed shape fills. For example, imagine a rectangle shape with the closed-frame shape fill and a pen width of 20. If you patterned this rectangle, QuickDraw GX would fill the frame of the rectangle with the pattern. See the section “Adding a Pattern to a Shape” on page 3-86 for examples.

QuickDraw GX uses the pattern property of a shape’s style object to store information about how to pattern the shape.

You use the pattern structure when specifying pattern information (using the `GXSetStylePattern` or `GXSetShapePattern` functions) and when retrieving pattern information (using the `GXGetStylePattern` or `GXGetShapePattern` functions).

The pattern structure is defined by the `gxPatternRecord` data type:

```
struct gxPatternRecord {
    gxPatternAttribute attributes;
    gxShape              pattern;
    gxPoint              u;
    gxPoint              v;
};
```

Field descriptions

<code>attributes</code>	Modifies the behavior of the pattern. The next section, “Pattern Attributes,” describes the <code>gxPatternAttribute</code> flags in detail.
<code>pattern</code>	Specifies the shape that makes up the pattern. You must use shapes in their primitive form for the pattern shape. (Primitive shapes are described in detail in Chapter 4, “Geometric Operations,” in this book.) You may not use text shapes, layout shapes, or picture shapes as the pattern shape. However, you may use framed shape shapes and shapes with an inverse shape fill. You may also use bitmap shapes with any pixel size as long as the bitmap shape does not contain color profile information. QuickDraw GX considers only the geometric properties (the shape type, the shape fill, and the shape geometry) of the shape specified by the pattern field. QuickDraw GX ignores the owner count, shape tags, and shape attributes properties and the style, ink, and transform objects of the pattern shape.

Geometric Styles

u	One of a pair of vectors that determine how QuickDraw GX places the pattern shape. This field, along with the v field, defines the pattern grid.
v	The other of the pair of vectors that describe how QuickDraw GX places the pattern shape. This field, along with the u field, defines the pattern grid.

The u and v fields together form a pair of vectors that define the pattern grid, which determines where QuickDraw GX places the pattern shape. The vectors define a grid of parallelograms and QuickDraw GX draws a pattern shape at every intersection in this grid.

The vectors specified by the u and v fields do not need to be any order, but they must point in different directions—that is, they may not lie on the same line. If you specify u and v vectors that are parallel, a `pattern_lattice_out_of_range` error results.

Optimization Note

QuickDraw GX draws bitmap patterns very quickly—that is, nearly as fast as a nonpatterned fill—if the u and v vectors place the patterns in a rectangular grid the size of the bitmap. ♦

See “Patterns” beginning on page 3-31 for more information about patterns and the pattern grid, and “Adding a Pattern to a Shape” on page 3-86 for an example of using patterns.

Pattern Attributes

Each pattern structure contains a set of flags that modify the way a shape is patterned. These constants are defined in the `gxPatternAttributes` enumeration:

```
enum gxPatternAttributes {
    gxPortAlignPattern = 0x0001,
    gxPortMapPattern   = 0x0002
};

typedef long gxPatternAttribute;
```

Constant descriptions**gxPortAlignPattern**

Indicates that QuickDraw GX should align the pattern shapes with the view device instead of the patterned shape. When this attribute is set, the pattern does not move when the patterned shape moves. Instead, the position of the pattern stays constant with respect to the view device. In effect, the patterned shape allows you to see through to a constant background covered by the pattern shape.

Geometric Styles

`gxPortMapPattern`

Indicates that mappings in the patterned shape's transform affect the patterned shape but do not affect the pattern. As an example, imagine that the transform of the patterned shape specifies that the patterned shape be scaled up by a factor of 2. If the `gxPortMapPattern` attribute is not set, then the pattern itself is magnified as well as the patterned shape. If this attribute is set, then the pattern stays the same size, but the patterned shape shows more of the pattern.

See the section “Patterns” on page 3-31 for an example of these attributes.

Functions

This section describes the functions available for manipulating a style object's geometric properties:

- the style attributes
- the curve error
- the pen width
- the caps
- the join
- the dash
- the pattern

These properties together determine the stylistic variations applied to the frame and the area of a shape when drawn.

For information about creating, disposing of, copying, and comparing style objects as well as information about manipulating style tags and style owner counts, see *Inside Macintosh: QuickDraw GX Objects*.

For information about the typographic style properties, such as font, text size, and text face, see *Inside Macintosh: QuickDraw GX Typography*.

In general, there are two types of functions that manipulate the properties of style objects:

- functions that require you to provide a reference to the style object itself
- functions that allow you to provide a reference to a shape and affect the style object associated with that shape

The section “Associating Styles With Shapes” on page 3-36 provides an example of both of these types of functions and compares their results.

Both types of functions are described in this reference section.

Getting and Setting Style Attributes

The style attributes are a set of flags that modify the behavior of the style object. In particular, these flags allow you to specify how QuickDraw GX places the geometric pen with respect to a shape's geometry and whether the shape should be constrained to a grid.

For a description of the style attributes, see the section “Style Attributes” on page 3-98.

You can use the `GXGetStyleAttributes` function to find the style attributes of an existing style and the `GXSetStyleAttributes` function to set the style attributes of a style.

The `GXGetShapeStyleAttributes` and `GXSetShapeStyleAttributes` functions provide a way to determine and change the style attributes of a style object associated with a particular shape.

GXGetStyleAttributes

You can use the `GXGetStyleAttributes` function to determine which style attributes are set for a particular style object.

```
gxStyleAttribute GXGetStyleAttributes(gxStyle source);
```

source A reference to the style object whose style attributes you want to determine.

function result The style attributes associated with the source style object.

DESCRIPTION

The `GXGetStyleAttributes` function returns as its function result the style attributes of the style object specified by the `source` parameter.

As an example, to examine the `gxSourceGridStyle` flag of a style object referenced by the variable `source`, you could use this code:

```
if (GXGetStyleAttributes(source) & gxSourceGridStyle) {
    /* style has gxSourceGridStyle attribute set */
}
```

Geometric Styles

The `gxCenterFrameStyle` attribute is set only if both the `gxInsideFrameStyle` and the `gxOutsideFrameStyle` attributes are clear, so if you want to test for a centered frame style you need this code:

```
if (GXGetStyleAttributes(source) &
    (gxInsideFrameStyle | gxOutsideFrameStyle) ==
    gxCenterFrameStyle) {
    /* style has gxCenterFrameStyle attribute set */
}
```

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

SEE ALSO

For a discussion of style attributes, see “Style Attributes” on page 3-98.

For an example of pen placement, see “Manipulating Pen Width and Placement” on page 3-51.

For an example of constraining shapes to grids, see “Constraining Shape Geometries to Grids” on page 3-40 and “Constraining Shapes to Device Grids” on page 3-42.

To examine the style attributes of a style object associated with a particular shape, use the `GXGetShapeStyleAttributes` function, which is described on page 3-112.

To alter the style attributes of a style object, use the `GXSetStyleAttributes` function, which is described in the next section.

To alter the style attributes of a style object associated with a particular shape, use the `GXSetShapeStyleAttributes` function, which is described on page 3-113.

GXSetStyleAttributes

You can use the `GXSetStyleAttributes` function to alter the style attributes for a particular style object.

```
void GXSetStyleAttributes(gxStyle target,
                          gxStyleAttribute attributes);
```

`target` A reference to the style object whose attributes you want to alter.

`attributes` The new set of attributes.

DESCRIPTION

The `GXSetStyleAttributes` function sets the style attributes of the style object specified by the `target` parameter to be the attributes specified in the `attributes` parameter.

You can use this function in combination with the `GXGetStyleAttributes` function to set or clear single style attributes. For example, to clear the `gxSourceGridStyle` attribute of a style object referenced by the variable `target`, you could use this line of code:

```
GXSetStyleAttributes(target,
                     GXGetStyleAttributes(target & ~gxSourceGridStyle);
```

To set the `gxSourceGridStyle` attribute, you could use this line of code:

```
GXSetStyleAttributes(target,
                     GXGetStyleAttributes(target | gxSourceGridStyle);
```

To set the `gxCenterFrameStyle` attribute, you need to clear the `gxInsideFrameStyle` and `gxOutsideFrameStyle` attributes.

When you set a style's attributes using this function, you are effectively changing the style attributes for all shapes that share the style.

ERRORS, WARNINGS, AND NOTICES**Errors**

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

Notices (debugging version)

<code>attributes_already_set</code>

SEE ALSO

For a discussion of style attributes, see “Style Attributes” on page 3-98.

For an example of pen placement, see “Manipulating Pen Width and Placement” beginning on page 3-51.

For an example of constraining shapes to grids, see “Constraining Shape Geometries to Grids” on page 3-40 and “Constraining Shapes to Device Grids” on page 3-42.

To examine the style attributes of a style object, use the `GXGetStyleAttributes` function, which is described on page 3-109.

To examine the style attributes of a style object associated with a particular shape, use the `GXGetShapeStyleAttributes` function, which is described in the next section. To alter the style attributes of a style object associated with a particular shape, use the `GXSetShapeStyleAttributes` function, which is described on page 3-113.

GXGetShapeStyleAttributes

You can use the `GXGetShapeStyleAttributes` function to determine which style attributes are set for the style object of a particular shape.

```
gxStyleAttribute GXGetShapeStyleAttributes(gxShape source);
```

`source` A reference to the shape whose style attributes you want to determine.

function result The style attributes of the source shape's style object.

DESCRIPTION

The `GXGetShapeStyleAttributes` function provides a convenient way to determine the style attributes of a shape without having to call the `GXGetShapeStyle` function to obtain a reference to the shape's style object.

As an example, to examine the `gxSourceGridStyle` flag of a style object associated with the shape object referenced by the variable `source`, you could use this code:

```
if (GXGetShapeStyleAttributes(source) & gxSourceGridStyle) {
    /* shape's style has gxSourceGridStyle attribute set */
}
```

The `gxCenterFrameStyle` attribute is set only if both the `gxInsideFrameStyle` and the `gxOutsideFrameStyle` attributes are clear, so if you want to test for a centered frame style you need this code:

```
if (GXGetShapeStyleAttributes(source) &
    (gxInsideFrameStyle | gxOutsideFrameStyle) ==
    gxCenterFrameStyle) {
    /* shape's style has gxCenterFrameStyle attribute set */
}
```

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

SEE ALSO

For a discussion of style attributes, see “Style Attributes” on page 3-98.

For an example of pen placement, see “Manipulating Pen Width and Placement” beginning on page 3-51.

Geometric Styles

For an example of constraining shapes to grids, see “Constraining Shape Geometries to Grids” on page 3-40 and “Constraining Shapes to Device Grids” on page 3-42.

To examine or alter the style attributes of a style object, use the `GXGetStyleAttributes` function, which is described on page 3-109. To alter the style attributes of a style object, use the `GXSetStyleAttributes` function, which is described on page 3-110.

To alter the style attributes of a style object associated with a particular shape, use the `GXSetShapeStyleAttributes` function, which is described in the next section.

GXSetShapeStyleAttributes

You can use the `GXSetShapeStyleAttributes` function to alter the style attributes of the style object associated with a particular shape.

```
void GXSetShapeStyleAttributes(gxShape target,
                               gxStyleAttribute attributes);
```

target A reference to the shape whose style attributes you want to alter.

attributes The new set of attributes.

DESCRIPTION

The `GXSetShapeStyleAttributes` function sets the style attributes of the style object associated with the shape specified by the `target` parameter.

If the target shape shares its style object with other shapes, this function makes a copy of the style object, sets the target shape to reference the copy, and changes the style attributes of the copy. (However, if the effect of this function would leave the style attributes unchanged, this function does not create a copy of the style object; instead, it posts a notice).

You can use this function in combination with the `GXGetShapeStyleAttributes` function to set or clear single style attributes. For example, to clear the `gxSourceGridStyle` attribute of a style object associated with a target shape, you could use this line of code:

```
GXSetShapeStyleAttributes(target,
                           GXGetShapeStyleAttributes(target & ~gxSourceGridStyle);
```

To set the `gxSourceGridStyle` attribute, you could use this line of code:

```
GXSetShapeStyleAttributes(target,
                           GXGetShapeStyleAttributes(target | gxSourceGridStyle);
```

Geometric Styles

ERRORS, WARNINGS, AND NOTICES

Errors

```

out_of_memory
style_is_nil
inconsistent_parameters    (debugging version)
parameter_out_of_range     (debugging version)

```

Notices (debugging version)

```

attributes_already_set

```

SEE ALSO

For a discussion of style attributes, see “Style Attributes” on page 3-98.

For an example of pen placement, see “Manipulating Pen Width and Placement” on page 3-51.

For an example of constraining shapes to grids, see “Constraining Shape Geometries to Grids” on page 3-40 and “Constraining Shapes to Device Grids” on page 3-42.

To examine the style attributes of a style object associated with a particular shape, use the `GXGetShapeStyleAttributes` function, which is described on page 3-112.

To examine the style attributes of a style object, use the `GXGetStyleAttributes` function, which is described on page 3-109. To alter the style attributes of a style object, use the `GXSetStyleAttributes` function, which is described on page 3-110.

Getting and Setting Curve Error

The curve error property of style objects specifies the allowable amount of error when QuickDraw GX converts a path shape into a polygon shape. It also specifies how far apart geometric points must be for QuickDraw GX to consider them separate points when performing geometric operations on shapes or reducing shapes.

For example, when you call the `GXInsetShape` function on a tight curve, the resulting curve can require many more geometric points than the original curve. QuickDraw GX simplifies the resulting shape by removing geometric points that are within the shape’s curve error from another geometric point.

You can use the `GXGetStyleCurveError` function to determine the curve error of a style object and the `GXSetStyleCurveError` function to change the curve error of a style object.

The `GXGetShapeCurveError` and `GXSetShapeCurveError` functions provide a way to determine and change the curve error of the style object associated with a particular shape.

GXGetStyleCurveError

You can use the `GXGetStyleCurveError` function to determine the curve error of a style object.

```
Fixed GXGetStyleCurveError(gxStyle source);
```

`source` A reference to the style object whose curve error you want to determine.

function result The curve error of the source style object.

DESCRIPTION

When a path shape has a curve error of 0, QuickDraw GX does not approximate the path shape with a polygon shape when converting it to a polygon. Instead, QuickDraw GX simply removes off-curve control points, as shown in “Using Curve Error When Converting Paths to Polygons” on page 3-45.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

SEE ALSO

For a discussion of curve error, see “Curve Error” on page 3-14.

For examples of using curve error, see “Using Curve Error When Converting Paths to Polygons” on page 3-45 and “Using Curve Error When Reducing Shapes” on page 3-49.

To change the curve error of a style object, use the `GXSetStyleCurveError` function, which is described in the next section.

To examine the curve error of a style object associated with a particular shape, use the `GXGetShapeCurveError` function, which is described on page 3-117. To change the curve error of a style object associated with a particular shape, use the `GXSetShapeCurveError` function, which is described on page 3-118.

GXSetStyleCurveError

You can use the `GXSetStyleCurveError` function to change the curve error of a style object.

```
void GXSetStyleCurveError(gxStyle target, Fixed error);
```

`target` A reference to the style object whose curve error you want to change.

`error` The new curve error.

DESCRIPTION

This routine sets the curve error of the style object specified by the `target` parameter to be the fixed-point value specified by the `error` parameter. You may specify any nonnegative value for this parameter.

When a path shape has a curve error of 0.0, QuickDraw GX does not approximate the path shape with a polygon shape when converting it to a polygon. Instead, QuickDraw GX simply removes off-curve control points, as shown in “Using Curve Error When Converting Paths to Polygons” on page 3-45.

A very small curve error may cause the `GXSetShapeType` function and certain geometric operations such as the `GXInsetShape` function to use inappropriate amounts of memory and time.

When you set a style’s curve error using this function, you are effectively changing the curve error for all shapes that share the style.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`style_is_nil`

`parameter_out_of_range` (debugging version)

Notices (debugging version)

`curve_error_already_set`

SEE ALSO

For a discussion of curve error, see “Curve Error” on page 3-14.

For examples of curve error, see “Using Curve Error When Converting Paths to Polygons” on page 3-45 and “Using Curve Error When Reducing Shapes” on page 3-49.

To determine the curve error of a style object, use the `GXGetStyleCurveError` function, which is described on page 3-115.

Geometric Styles

To examine the curve error of a style object associated with a particular shape, use the `GXGetShapeCurveError` function, which is described in the next section. To change the curve error of a style object associated with a particular shape, use the `GXSetShapeCurveError` function, which is described on page 3-118.

GXGetShapeCurveError

You can use the `GXGetShapeCurveError` function to determine the curve error of the style object associated with a particular shape.

```
Fixed GXGetShapeCurveError(gxShape source);
```

`source` A reference to the shape whose curve error you want to determine.

function result The curve error of the style object associated with the source shape.

DESCRIPTION

When a path shape has a curve error of 0, QuickDraw GX does not approximate the path shape with a polygon shape when converting it to a polygon. Instead, QuickDraw GX simply removes off-curve control points, as shown in “Using Curve Error When Converting Paths to Polygons” on page 3-45.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

SEE ALSO

For a discussion of curve error, see “Curve Error” on page 3-14.

For examples of curve error, see “Using Curve Error When Converting Paths to Polygons” on page 3-45 and “Using Curve Error When Reducing Shapes” on page 3-49.

To determine the curve error of a style object, use the `GXGetStyleCurveError` function, which is described on page 3-115. To change the curve error of a style object, use the `GXSetStyleCurveError` function, which is described on page 3-116.

To change the curve error of a style object associated with a particular shape, use the `GXSetShapeCurveError` function, which is described in the next section.

GXSetShapeCurveError

You can use the `GXSetShapeCurveError` function to change the curve error of the style object associated with a particular shape.

```
void GXSetShapeCurveError(gxShape target, Fixed error);
```

`target` A reference to the shape whose curve error you want to change.

`error` The new curve error.

DESCRIPTION

The `GXSetShapeCurveError` function replaces the curve error of the style object associated with the shape specified by the `source` parameter with the value in the `error` parameter. You may specify any nonnegative value for this parameter.

If the target shape shares its style object with other shapes, this function makes a copy of the style object, sets the target shape to reference the copy, and changes the curve error of the copy. (However, if the effect of this function would leave the curve error unchanged, this function does not create a copy of the style object; instead, it posts a notice.)

When the curve error is 0, QuickDraw GX does not approximate a path shape with a polygon shape when converting from a path to a polygon. Instead, QuickDraw GX simply removes off-curve control points, as shown in “Using Curve Error When Converting Paths to Polygons” on page 3-45.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`shape_is_nil`

`parameter_out_of_range` (debugging version)

Notices (debugging version)

`curve_error_already_set`

SEE ALSO

For a discussion of curve error, see “Curve Error” on page 3-14.

For examples of curve error, see “Using Curve Error When Converting Paths to Polygons” on page 3-45 and “Using Curve Error When Reducing Shapes” on page 3-49.

To determine the curve error of a style object, use the `GXGetStyleCurveError` function, which is described on page 3-115. To change the curve error of a style object, use the `GXSetStyleCurveError` function, which is described on page 3-116.

To determine the curve error of a style object associated with a particular shape, use the `GXGetShapeCurveError` function, which is described on page 3-117.

Getting and Setting the Pen Width

The pen width property of a style object specifies the width at which QuickDraw GX should draw a shape's contours. A pen width of 0 specifies a hairline. QuickDraw GX always draws hairlines at the resolution of the output device—one pixel wide. The pen width also affects dashing: QuickDraw GX scales a shape's dashes (in the y-coordinate direction) by the pen width. Also, the pen width affects the clip shape that QuickDraw GX uses to clip the dashes when a shape's clip dash attribute is set.

You can use the `GXGetStylePen` function to determine the pen width of a style object and the `GXSetStylePen` function to change the pen width of a style object.

The `GXGetShapePen` and `GXSetShapePen` functions provide a way to determine and change the pen width of the style object associated with a particular shape.

GXGetStylePen

You can use the `GXGetStylePen` function to determine the pen width of a particular style object.

```
Fixed GXGetStylePen(gxStyle source);
```

source A reference to the style object whose pen width you want to determine.

function result The pen width of the source style object.

DESCRIPTION

A pen width of 0.0 indicates a hairline width; QuickDraw GX always draws hairlines one pixel wide.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

SEE ALSO

For a discussion of the drawing pen, see “The Geometric Pen” on page 3-15.

For an example of changing a shape's pen width, see “Manipulating Pen Width and Placement” on page 3-51.

To change the pen width of a style object, use the `GXSetStylePen` function, which is described in the next section.

Geometric Styles

To determine the pen width of a style object associated with a particular shape, use the `GXGetShapePen` function, which is described on page 3-121. To change the pen width of a style object associated with a particular shape, use the `GXSetShapePen` function, which is described on page 3-122.

GXSetStylePen

You can use the `GXSetStylePen` function to change the pen width of a style object.

```
void GXSetStylePen(gxStyle target, Fixed pen);
```

`target` A reference to the style object whose pen width you want to change.

`pen` The new pen width.

DESCRIPTION

The `GXSetStylePen` function sets the pen width of the style object specified by the `target` parameter to the value specified in the `pen` parameter. You may specify any nonnegative value for this parameter.

A pen width of 0 indicates a hairline; QuickDraw GX always draws hairlines one pixel wide.

Remember that the `pen` parameter is specified as a fixed-point value. Very small diameters may cause all drawing to disappear, since a shape may fall between pixels. A common mistake when setting the pen width is to specify the pen width as an integer, rather than a fixed-point value:

```
GXSetStylePen(myStyle, 1); /* set the pen width to 1/65536 */
```

```
GXSetStylePen(myStyle, ff(1)); /* set the pen width to 1.0 */
```

When you set the pen width using this function, you are effectively changing the pen width for all shapes that share the style.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`style_is_nil`

`parameter_out_of_range` (debugging version)

Notices (debugging version)

`pen_size_already_set`

SEE ALSO

For a discussion of the drawing pen, see “The Geometric Pen” on page 3-15.

For an example of changing a shape’s pen width, see “Manipulating Pen Width and Placement” on page 3-51.

To determine the pen width of a style object, use the `GXGetStylePen` function, which is described on page 3-119.

To determine the pen width of a style object associated with a particular shape, use the `GXGetShapePen` function, which is described in the next section. To change the pen width of a style object associated with a particular shape, use the `GXSetShapePen` function, which is described on page 3-122.

GXGetShapePen

You can use the `GXGetShapePen` function to determine the pen width of the style object associated with a particular shape.

```
Fixed GXGetShapePen(gxShape source);
```

`source` A reference to the shape whose pen width you want to determine.

function result The pen width of the source shape’s style object.

DESCRIPTION

A pen width of 0.0 indicates a hairline width; QuickDraw GX always draws hairlines one pixel wide.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

SEE ALSO

For a discussion of the drawing pen, see “The Geometric Pen” on page 3-15.

For an example of changing a shape’s pen width, see “Manipulating Pen Width and Placement” on page 3-51.

Geometric Styles

To determine the pen width of a style object, use the `GXGetStylePen` function, which is described on page 3-119. To change the pen width of a style object, use the `GXSetStylePen` function, which is described on page 3-120.

To change the pen width of a style object associated with a particular shape, use the `GXSetShapePen` function, which is described in the next section.

GXSetShapePen

You can use the `GXSetShapePen` function to change the pen width of the style object associated with a particular shape.

```
void GXSetShapePen(gxShape target, Fixed pen);
```

`target` A reference to the shape whose pen width you want to change.

`pen` The new pen width.

DESCRIPTION

The `GXSetShapePen` function sets the pen width of the target shape's style object to be the value specified in the `pen` parameter. You may specify any nonnegative value for this parameter.

If the target shape shares its style object with other shapes, this function makes a copy of the style object, sets the target shape to reference the copy, and changes the pen width of the copy. (However, if the effect of this function would leave the pen width information unchanged, this function does not create a copy of the style object; instead, it posts a notice.)

A pen width of 0 indicates a hairline; QuickDraw GX always draws hairlines one pixel wide.

```
GXSetShapePen(myShape, 0); /* set as thin as renderable */
```

Remember that the `pen` parameter is specified as a fixed-point value. Very small diameters may cause all drawing to disappear, since a shape may fall between pixels. A common mistake when setting the pen width is to specify the pen width as an integer, rather than a fixed-point value:

```
GXSetStylePen(myStyle, 1); /* set the pen width to 1/65536 */
```

```
GXSetStylePen(myStyle, ff(1)); /* set the pen width to 1.0 */
```

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`parameter_out_of_range` (debugging version)

Notices (debugging version)

`pen_size_already_set`

SEE ALSO

For a discussion of the drawing pen, see “The Geometric Pen” on page 3-15.

For an example of changing a shape’s pen width, see “Manipulating Pen Width and Placement” on page 3-51.

To determine the pen width of a style object, use the `GXGetStylePen` function, which is described on page 3-119. To change the pen width of a style object, use the `GXSetStylePen` function, which is described on page 3-120.

To determine the pen width of a style object associated with a particular shape, use the `GXGetShapePen` function, which is described on page 3-121.

Getting and Setting Caps

QuickDraw GX allows you to specify what to draw at the start and at the end of a shape’s contours. In particular, you may specify a start cap for any point shape, and you may specify a start cap and an end cap for any line, curve, polygon, or path shape that has an `gxOpenFrameFill` shape fill. You must always specify cap shapes in primitive form.

“The Cap Structure” on page 3-99 describes the `gxCapRecord` structure, which you use when retrieving or specifying cap information. That section also describes what types of shapes you may use as cap shapes.

You can use the `GXGetStyleCap` function to retrieve the cap information from a style object and the `GXSetStyleCap` function to specify cap information for a style object.

The `GXGetShapeCap` and `GXSetShapeCap` functions provide a way to retrieve and specify cap information for the style object associated with a particular shape.

GXGetStyleCap

You can use the `GXGetStyleCap` function to retrieve the cap information from a style object.

```
gxCapRecord *GXGetStyleCap(gxStyle source, gxCapRecord *cap);
```

source The style object whose cap information you want to retrieve.

cap A pointer to a `gxCapRecord` structure. On return, this structure contains the cap information for the source style object.

function result A copy of the `gxCapRecord` associated with the source style.

DESCRIPTION

The `GXGetStyleCap` function returns as its function result, and in the `cap` parameter, a `gxCapRecord` structure containing the cap information for the style object specified by the `source` parameter.

This function creates new shapes to encapsulate the start cap and end cap geometries, and places references to these shapes in the `startCap` and `endCap` fields of the returned `gxCapRecord` structure. You should dispose of these shapes when you no longer need them.

Since this function copies the cap information from the source style object, you may make changes to the `gxCapRecord` structure returned by this function without affecting the source style's cap information. If you want to change the cap information in the source style, you must use the `GXSetStyleCap` function.

SPECIAL CONSIDERATIONS

If no error results, the `GXGetStyleCap` function creates shapes; you are responsible for disposing of these shapes when you no longer need them. See *Inside Macintosh: QuickDraw GX Objects* for information about disposing QuickDraw GX objects.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`
`parameter_is_nil` (debugging version)

SEE ALSO

For a discussion of start and end caps, see “Caps” on page 3-23.

For examples of adding caps to a shape, see “Adding Caps to a Shape” on page 3-57 and “Adding Standard Caps to a Shape” on page 3-59.

For a discussion of the `gxCapRecord` structure and a description of what types of shapes you can use as cap shapes, see “The Cap Structure” on page 3-99.

To specify cap information for a style object, use the `GXSetStyleCap` function, which is described in the next section.

To retrieve cap information from a style object associated with a particular shape, use the `GXGetShapeCap` function, which is described on page 3-126. To specify cap information for a style object associated with a particular shape, use the `GXSetShapeCap` function, which is described on page 3-128.

GXSetStyleCap

You can use the `GXSetStyleCap` function to change the cap information of a style object.

```
void GXSetStyleCap(gxStyle target, const gxCapRecord *cap);
```

`target` The style object whose cap information you want to change.

`cap` A pointer to the new cap information.

DESCRIPTION

The `GXSetStyleCap` function replaces the cap information in the style object specified by the `target` parameter with the cap information specified in the `cap` parameter. You use the `gxCapRecord` structure to provide cap information.

Passing `nil` for the `cap` parameter indicates that you want no caps and QuickDraw GX removes any cap information from the target style.

When you set a style’s cap property using this function, you are effectively changing the caps for all shapes that share the style.

Geometric Styles

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>style_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>empty_shape_not_allowed</code>	(debugging version)
<code>ignorePlatformShape_not_allowed</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)
<code>nil_style_in_glyph_not_allowed</code>	(debugging version)
<code>complex_glyph_style_not_allowed</code>	(debugging version)
<code>shapeFill_not_allowed</code>	(debugging version)

Notices (debugging version)

<code>caps_already_set</code>
<code>tags_in_shape_ignored</code>

SEE ALSO

For a discussion of start and end caps, see “Caps” on page 3-23.

For examples of adding caps to a shape, see “Adding Caps to a Shape” on page 3-57 and “Adding Standard Caps to a Shape” on page 3-59.

For a discussion of the `gxCapRecord` structure and a description of what types of shapes you can use as cap shapes, see “The Cap Structure” on page 3-99.

To retrieve cap information from a style object, use the `GXGetStyleCap` function, which is described on page 3-124.

To retrieve cap information from a style object associated with a particular shape, use the `GXGetShapeCap` function, which is described in the next section. To specify cap information for a style object associated with a particular shape, use the `GXSetShapeCap` function, which is described on page 3-128.

GXGetShapeCap

You can use the `GXGetShapeCap` function to retrieve cap information from the style object of a particular shape.

```
gxCapRecord *GXGetShapeCap(gxShape source, gxCapRecord *cap);
```

source A reference to the shape whose cap information you want to retrieve.

cap A pointer to a `gxCapRecord` structure. On return, this structure contains the cap information for the source shape.

function result A copy of the `gxCapRecord` structure associated with the source shape’s style object.

DESCRIPTION

The `GXGetShapeCap` function returns as its function result, and in the `cap` parameter, a `gxCapRecord` structure containing the cap information for the style object associated with the shape specified by the source parameter.

This function creates new shapes to encapsulate the start cap and end cap geometries, and places references to these shapes in the `startCap` and `endCap` fields of the returned `gxCapRecord` structure. You should dispose of these shapes when you no longer need them.

Since this function copies the cap information from the source shape's style, you may make changes to the `gxCapRecord` structure returned by this function without affecting the source shape's caps. If you want to change the cap information for the source shape, you must use the `GXSetShapeCap` function.

SPECIAL CONSIDERATIONS

Unless an error results, the `GXGetShapeCap` function creates shapes; you are responsible for disposing of these shapes when you no longer need them. See *Inside Macintosh: QuickDraw GX Objects* for information about disposing of QuickDraw GX objects.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`shape_is_nil`
`parameter_is_nil`

SEE ALSO

For a discussion of start and end caps, see “Caps” on page 3-23.

For examples of adding caps to a shape, see “Adding Caps to a Shape” on page 3-57 and “Adding Standard Caps to a Shape” on page 3-59.

For a discussion of the `gxCapRecord` structure and a description of what types of shapes you can use as cap shapes, see “The Cap Structure” on page 3-99.

To retrieve cap information from a style object, use the `GXGetStyleCap` function, which is described on page 3-124. To specify cap information for a style object, use the `GXSetStyleCap` function, which is described on page 3-125.

To specify cap information for a style object associated with a particular shape, use the `GXSetShapeCap` function, which is described in the next section.

GXSetShapeCap

You can use the `GXSetShapeCap` function to change the cap information of the style object associated with a particular shape.

```
void GXSetShapeCap(gxShape target, const gxCapRecord *cap);
```

`target` A reference to the shape whose cap information you want to change.

`cap` A pointer to the new cap information.

DESCRIPTION

The `GXSetShapeCap` function replaces the cap information in the style object of the shape specified by the `target` parameter with the cap information specified in the `cap` parameter. You use the `gxCapRecord` structure to provide cap information.

Passing `nil` for the `cap` parameter indicates that you want no caps and QuickDraw GX removes any cap information from the target shape.

If the target shape shares its style object with other shapes, this function makes a copy of the style object, sets the target shape to reference the copy, and changes the cap property of the copy. (However, if the effect of this function would leave the cap information unchanged, this function does not create a copy of the style object; instead, it posts a notice.)

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>style_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>empty_shape_not_allowed</code>	(debugging version)
<code>ignorePlatformShape_not_allowed</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)
<code>nil_style_in_glyph_not_allowed</code>	(debugging version)
<code>complex_glyph_style_not_allowed</code>	(debugging version)
<code>shapeFill_not_allowed</code>	(debugging version)

Notices (debugging version)

`caps_already_set`
`tags_in_shape_ignored`

SEE ALSO

For a discussion of start and end caps, see “Caps” on page 3-23.

For examples of adding caps to shapes, see “Adding Caps to a Shape” on page 3-57 and “Adding Standard Caps to a Shape” on page 3-59.

For a discussion of the `gxCapRecord` structure and a description of what types of shapes you can use as cap shapes, see “The Cap Structure” on page 3-99.

To retrieve cap information from a style object, use the `GXGetStyleCap` function, which is described on page 3-124.

To specify cap information for a style object, use the `GXSetStyleCap` function, which is described on page 3-125.

To retrieve cap information from a style object associated with a particular shape, use the `GXGetShapeCap` function, which is described on page 3-126.

Getting and Setting Joins

QuickDraw GX allows you to specify what to draw at corners of a shape's contours. In particular, you may specify a corner join for any rectangle, polygon, or path shape that has an open-frame shape fill or a closed-frame shape fill. You must always specify join shapes in primitive form.

"The Join Structure" on page 3-101 describes the `gxCapRecord` structure, which you use when retrieving or specifying join information. That section also describes what types of shapes you may use as join shapes.

You can use the `GXGetStyleJoin` function to retrieve the join information from a style object and the `GXSetStyleJoin` function to specify join information for a style object.

The `GXGetShapeJoin` and `GXSetShapeJoin` functions provide a way to retrieve and specify join information for the style object associated with a particular shape.

GXGetStyleJoin

You can use the `GXGetStyleJoin` function to retrieve the join information from a style object.

```
gxJoinRecord *GXGetStyleJoin(gxStyle source, gxJoinRecord *join);
```

source A reference to the style object whose join information you want to retrieve.

join A pointer to a `gxJoinRecord` structure. On return, this structure contains the join information for the source style object.

function result A copy of the `gxJoinRecord` structure associated with the source style object.

DESCRIPTION

The `GXGetStyleJoin` function returns as its function result, and in the `join` parameter, a pointer to a `gxJoinRecord` structure containing the join information for the style object specified by the `source` parameter.

Geometric Styles

This function creates a new shape to encapsulate the join geometry, and places a reference to this shape in the `join` field of the returned `gxJoinRecord` structure. You should dispose of this shape when you no longer need it.

Since this function copies the join information from the source style, you may make changes to the `gxJoinRecord` structure returned by this function without affecting the source style's join information. If you want to change the join information in the source style, you must use the `GXSetStyleJoin` function.

SPECIAL CONSIDERATIONS

Unless an error results, the `GXGetStyleJoin` function creates a shape; you are responsible for disposing of this shape when you no longer need it. See *Inside Macintosh: QuickDraw GX Objects* for information about disposing of QuickDraw GX objects.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`
`parameter_is_nil`

SEE ALSO

For a discussion of joins, see “Joins” on page 3-25.

For examples of adding joins to shapes, see “Adding Joins to a Shape” on page 3-61 and “Adding Standard Joins to a Shape” on page 3-64.

For a discussion of the `gxJoinRecord` structure and a description of what types of shapes you can use as join shapes, see “The Join Structure” on page 3-101.

To specify join information for a style object, use the `GXSetStyleJoin` function, which is described in the next section.

To retrieve join information from a style object associated with a particular shape, use the `GXGetShapeJoin` function, which is described on page 3-132.

To specify join information for a style object associated with a particular shape, use the `GXSetShapeJoin` function, which is described on page 3-133.

GXSetStyleJoin

You can use the `GXSetStyleJoin` function to change a style object's join information.

```
void GXSetStyleJoin(gxStyle target, const gxJoinRecord *join);
```

Geometric Styles

`target` A reference to the style object whose join information you want to change.
`join` A pointer to the new join information.

DESCRIPTION

The `GXSetStyleJoin` function replaces the join information in the style object specified by the `target` parameter with the join information specified in the `join` parameter. You use the `gxJoinRecord` structure to provide join information.

Passing `nil` for the `join` parameter indicates that you want no join shape and QuickDraw GX removes any join information from the target style.

When you set a style's join property using this function, you are effectively changing the joins for all shapes that share the style.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>style_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>empty_shape_not_allowed</code>	(debugging version)
<code>ignorePlatformShape_not_allowed</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)
<code>nil_style_in_glyph_not_allowed</code>	(debugging version)
<code>complex_glyph_style_not_allowed</code>	(debugging version)
<code>shapeFill_not_allowed</code>	(debugging version)

Notices (debugging version)

`join_type_already_set`
`tags_in_shape_ignored`

SEE ALSO

For a discussion of joins, see “Joins” on page 3-25.

For examples of adding joins to shapes, see “Adding Joins to a Shape” on page 3-61 and “Adding Standard Joins to a Shape” on page 3-64.

For a discussion of the `gxJoinRecord` structure and a description of what types of shapes you can use as join shapes, see “The Join Structure” on page 3-101.

To retrieve join information from a style object, use the `GXGetStyleJoin` function, which is described on page 3-129.

To retrieve join information from a style object associated with a particular shape, use the `GXGetShapeJoin` function, which is described in the next section.

To specify join information for a style object associated with a particular shape, use the `GXSetShapeJoin` function, which is described on page 3-133.

GXGetShapeJoin

You can use the `GXGetShapeJoin` function to retrieve the join information from the style object of a shape.

```
gxJoinRecord *GXGetShapeJoin(gxShape source, gxJoinRecord *join);
```

`source` A reference to the shape whose join information you want to retrieve.

`join` A pointer to a `gxJoinRecord` structure. On return, this structure contains the join information for the source shape.

function result A copy of the `gxJoinRecord` structure associated with the source shape's style object.

DESCRIPTION

The `GXGetShapeJoin` function returns as its function result, and in the `join` parameter, a pointer to a `gxJoinRecord` structure containing the join information for the style object of the shape specified by the `source` parameter.

This function creates a new shape to encapsulate the join geometry, and places a reference to this shape in the `join` field of the returned `gxJoinRecord` structure. You should dispose of this shape when you no longer need it.

Since this function copies the join information from the source shape's style, you may make changes to the `gxJoinRecord` structure returned by this function without affecting the source shape's joins. If you want to change the join information for the source shape, you must use the `GXSetShapeJoin` function.

SPECIAL CONSIDERATIONS

Unless an error results, the `GXGetShapeJoin` function creates a shape; you are responsible for disposing of this shape when you no longer need it. See *Inside Macintosh: QuickDraw GX Objects* for information about disposing of QuickDraw GX objects.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`shape_is_nil`

`parameter_is_nil`

SEE ALSO

For a discussion of joins, see “Joins” on page 3-25.

For examples of adding joins to shapes, see “Adding Joins to a Shape” on page 3-61 and “Adding Standard Joins to a Shape” on page 3-64.

For a discussion of the `gxJoinRecord` structure and a description of what types of shapes you can use as join shapes, see “The Join Structure” on page 3-101.

To retrieve join information from a style object, use the `GXGetStyleJoin` function, which is described on page 3-129. To specify join information for a style object, use the `GXSetStyleJoin` function, which is described on page 3-130.

To specify join information for a style object associated with a particular shape, use the `GXSetShapeJoin` function, which is described in the next section.

GXSetShapeJoin

You can use the `GXSetShapeJoin` function to change the join information for the style object of a particular shape.

```
void GXSetShapeJoin(gxShape target, const gxJoinRecord *join);
```

`target` A reference to the shape whose join information you want to change.

`join` A pointer to new join information.

DESCRIPTION

The `GXSetShapeJoin` function replaces the join information in the style object of the shape specified by the `target` parameter with the join information provided in the `join` parameter. You use the `gxJoinRecord` structure to provide join information.

Passing `nil` for the `join` parameter indicates that you want no joins and QuickDraw GX removes any join information from the target shape.

If the target shape shares its style object with other shapes, this function makes a copy of the style object, sets the target shape to reference the copy, and changes the join property of the copy. (However, if the effect of this function would leave the join information unchanged, this function does not create a copy of the style object; instead, it posts a notice.)

Geometric Styles

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory	
style_is_nil	
parameter_out_of_range	(debugging version)
empty_shape_not_allowed	(debugging version)
ignorePlatformShape_not_allowed	(debugging version)
illegal_type_for_shape	(debugging version)
nil_style_in_glyph_not_allowed	(debugging version)
complex_glyph_style_not_allowed	(debugging version)
shapeFill_not_allowed	(debugging version)

Notices (debugging version)

join_type_already_set
tags_in_shape_ignored

SEE ALSO

For a discussion of joins, see “Joins” on page 3-25.

For examples of adding joins to shapes, see “Adding Joins to a Shape” on page 3-61 and “Adding Standard Joins to a Shape” on page 3-64.

For a discussion of the `gxJoinRecord` structure and a description of what types of shapes you can use as join shapes, see “The Join Structure” on page 3-101.

To retrieve join information from a style object, use the `GXGetStyleJoin` function, which is described on page 3-129. To specify join information for a style object, use the `GXSetStyleJoin` function, which is described on page 3-130.

To retrieve join information from a style object associated with a particular shape, use the `GXGetShapeJoin` function, which is described on page 3-132.

Getting and Setting Dashes

QuickDraw GX allows you to specify how contours of a shape should be dashed when drawn. In particular, you may specify a dash shape for any line, curve, rectangle, polygon, or path shape that has an open-frame shape fill or a closed-frame shape fill. You must always specify dash shapes in primitive form.

“The Dash Structure” on page 3-103 describes the `gxDashRecord` structure, which you use when retrieving or specifying dash information. That section also describes what types of shapes you may use as a dash shape.

You can use the `GXGetStyleDash` function to retrieve the dash information from a style object and the `GXSetStyleDash` function to specify dash information for a style object.

The `GXGetShapeDash` and `GXSetShapeDash` functions provide a way to retrieve and specify dash information for the style object associated with a particular shape.

GXGetStyleDash

You can use the `GXGetStyleDash` function to retrieve the dash information from a style object.

```
gxDashRecord *GXGetStyleDash(gxStyle source, gxDashRecord *dash);
```

source A reference to the style object whose dash information you want to retrieve.

dash A pointer to a `gxDashRecord` structure. On return, this structure contains the dash information for the source style object.

function result A copy of the `gxDashRecord` structure associated with the source style object.

DESCRIPTION

The `GXGetStyleDash` function returns as its function result, and in the `dash` parameter, a pointer to a `gxDashRecord` structure containing the dash information for the style object specified by the `source` parameter.

This function creates a new shape to encapsulate the dash geometry and places a reference to this shape in the `dash` field of the returned `gxDashRecord` structure. You should dispose of this shape when you no longer need it.

Since this function copies the dash information from the source style, you may make changes to the `gxDashRecord` structure returned by this function without affecting the source style's dash information. If you want to change the dash information in the source style, you must use the `GXSetStyleDash` function.

SPECIAL CONSIDERATIONS

Unless an error results, the `GXGetStyleDash` function creates a shape; you are responsible for disposing of this shape when you no longer need it. See *Inside Macintosh: QuickDraw GX Objects* for information about disposing of QuickDraw GX objects.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`
`parameter_is_nil`

Geometric Styles

SEE ALSO

For a discussion of dashes, see “Dashes” on page 3-27.

For examples of adding dashes to shapes, see page 3-66 through page 3-86.

For a discussion of the `gxDashRecord` structure and a description of what types of shapes you can use as dash shapes, see “The Dash Structure” on page 3-103.

To specify dash information for a style object, use the `GXSetStyleDash` function, which is described in the next section.

To retrieve dash information from a style object associated with a particular shape, use the `GXGetShapeDash` function, which is described on page 3-138. To specify dash information for a style object associated with a particular shape, use the `GXSetShapeDash` function, which is described on page 3-139.

To determine where dashing occurs for a particular shape, use the `GXGetShapeDashPositions` function, which is described on page 3-140.

GXSetStyleDash

You can use the `GXSetStyleDash` function to change a style object’s dash information.

```
void GXSetStyleDash(gxStyle target, const gxDashRecord *dash);
```

`target` A reference to the style object whose dash information you want to change.

`dash` A pointer to the new dash information.

DESCRIPTION

The `GXSetStyleDash` function replaces the dash information in the style object specified by the `target` parameter with the dash information provided by the `dash` parameter. You use the `gxDashRecord` structure to provide dash information.

Passing `nil` for the `dash` parameter indicates that you want no dashing to occur and QuickDraw GX removes any dash information from the target style.

When you set a style’s dash property using this function, you are effectively changing the dashes for all shapes that share the style.

Geometric Styles

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory	
style_is_nil	
parameter_out_of_range	(debugging version)
empty_shape_not_allowed	(debugging version)
ignorePlatformShape_not_allowed	(debugging version)
illegal_type_for_shape	(debugging version)
nil_style_in_glyph_not_allowed	(debugging version)
complex_glyph_style_not_allowed	(debugging version)
shapeFill_not_allowed	(debugging version)

Warnings

graphic_type_cannot_be_dashed

Notices (debugging version)

dash_already_set
tags_in_shape_ignored

SEE ALSO

For a discussion of dashes, see “Dashes” on page 3-27.

For examples of adding dashes to shapes, see page 3-66 through page 3-86.

For a discussion of the `gxDashRecord` structure and a description of what types of shapes you can use as dash shapes, see “The Dash Structure” on page 3-103.

To retrieve dash information from a style object, use the `GXGetStyleDash` function, which is described on page 3-135.

To retrieve dash information from a style object associated with a particular shape, use the `GXGetShapeDash` function, which is described in the next section. To specify dash information for a style object associated with a particular shape, use the `GXSetShapeDash` function, which is described on page 3-139.

To determine where dashing occurs for a particular shape, use the `GXGetShapeDashPositions` function, which is described on page 3-140.

GXGetShapeDash

You can use the `GXGetShapeDash` function to retrieve the dash information from the style object associated with a particular shape.

```
gxDashRecord *GXGetShapeDash(gxShape source, gxDashRecord *dash);
```

`source` A reference to the shape whose dash information you want to retrieve.

`dash` A pointer to a `gxDashRecord` structure. On return, this structure contains the dash information for the source shape.

function result A copy of the `gxDashRecord` structure associated with the source shape's style object.

DESCRIPTION

The `GXGetShapeDash` function returns as its function result and in the `dash` parameter, a pointer to a `gxDashRecord` structure containing the dash information for the style object of the shape specified by the `source` parameter.

This function creates a new shape to encapsulate the dash geometry, and places a reference to this shape in the `dash` field of the returned `gxDashRecord` structure. You should dispose of this shape when you no longer need it.

Since this function copies the dash information from the source shape's style, you may make changes to the `gxDashRecord` structure returned by this function without affecting the source shape's dashes. If you want to change the dash information for the source shape, you must use the `GXSetShapeDash` function.

SPECIAL CONSIDERATIONS

Unless an error results, the `GXGetShapeDash` function creates a shape; you are responsible for disposing of this shape when you no longer need it. See *Inside Macintosh: QuickDraw GX Objects* for information about disposing of objects.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`shape_is_nil`

`parameter_is_nil`

SEE ALSO

For a discussion of dashes, see “Dashes” on page 3-27.

For examples of adding dashes to shapes, see page 3-66 through page 3-86.

For a discussion of the `gxDashRecord` structure and a description of what types of shapes you can use as dash shapes, see “The Dash Structure” on page 3-103.

To retrieve dash information from a style object, use the `GXGetStyleDash` function, which is described on page 3-135.

To specify dash information for a style object, use the `GXSetStyleDash` function, which is described on page 3-136.

To specify dash information for a style object associated with a particular shape, use the `GXSetShapeDash` function, which is described in the next section.

To determine where dashing occurs for a particular shape, use the `GXGetShapeDashPositions` function, which is described on page 3-140.

GXSetShapeDash

You can use the `GXSetShapeDash` function to change the dash information for a style object associated with a particular shape.

```
void GXSetShapeDash(gxShape target, const gxDashRecord *dash);
```

target A reference to the shape whose dash information you want to change.

dash A pointer to the new dash information.

DESCRIPTION

The `GXSetShapeDash` function replaces the dash information in the style object of the shape specified by the `target` parameter with the dash information provided by the `dash` parameter. You use the `gxDashRecord` structure to provide dash information.

Passing `nil` for the `dash` parameter indicates that you want no dashing to occur and QuickDraw GX removes any dash information from the target shape.

If the target shape shares its style object with other shapes, this function makes a copy of the style object, sets the target shape to reference the copy, and changes the dash property of the copy. (However, if the effect of this function would leave the dash information unchanged, this function does not create a copy of the style object; instead, it returns a notice.)

Geometric Styles

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory	
style_is_nil	
parameter_out_of_range	(debugging version)
empty_shape_not_allowed	(debugging version)
ignorePlatformShape_not_allowed	(debugging version)
illegal_type_for_shape	(debugging version)
nil_style_in_glyph_not_allowed	(debugging version)
complex_glyph_style_not_allowed	(debugging version)
shapeFill_not_allowed	(debugging version)

Warnings

graphic_type_cannot_be_dashed

Notices (debugging version)

dash_already_set
tags_in_shape_ignored

SEE ALSO

For a discussion of dashes, see “Dashes” on page 3-27.

For examples of adding dashes to shapes, see page 3-66 through page 3-86.

For a discussion of the `gxDashRecord` structure and a description of what types of shapes you can use as dash shapes, see “The Dash Structure” on page 3-103.

To retrieve dash information from a style object, use the `GXGetStyleDash` function, which is described on page 3-135.

To specify dash information for a style object, use the `GXSetStyleDash` function, which is described on page 3-136.

To retrieve dash information from a style object associated with a particular shape, use the `GXGetShapeDash` function, which is described on page 3-138.

To determine where dashing occurs for a particular shape, use the `GXGetShapeDashPositions` function, which is described in the next section.

GXGetShapeDashPositions

You can use the `GXGetShapeDashPositions` function to determine the precise locations where QuickDraw GX draws a particular shape’s dashes.

```
long GXGetShapeDashPositions(gxShape source,
                             gxMapping dashMappings[]);
```


Geometric Styles

source A reference to the shape whose dash positions you want to find.

dashMappings An array of dash positions. On return, this array contains mappings that indicate the position of the dashes of the source shape.

function result The number of dash positions returned in the `dashMappings` parameter.

DESCRIPTION

The `GXGetShapeDashPositions` function returns in the `dashMappings` parameter mappings that indicate the locations and rotations of the dashes as drawn along the contours of the source shape.

The function result is the number of dash positions returned—the number of dash shapes drawn along the contours of the source shape.

If you pass `nil` for the `dashMappings` parameter, the `GXGetShapeDashPositions` function still returns as the function result the number of dashes but it does not return the positions of the dashes.

This function returns 0 if the source shape is not dashed.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

Warnings

`graphic_type_cannot_be_dashed`

SEE ALSO

For a discussion of dashes, see “Dashes” on page 3-27.

For an example of using this function, see “Determining Dash Positions” on page 3-81.

For a discussion of the `gxDashRecord` structure, see “The Dash Structure” on page 3-103.

To retrieve dash information from a style object, use the `GXGetStyleDash` function, which is described on page 3-135. To specify dash information for a style object, use the `GXSetStyleDash` function, which is described on page 3-136.

To retrieve dash information from a style object associated with a particular shape, use the `GXGetShapeDash` function, which is described on page 3-138. To specify dash information for a style object associated with a particular shape, use the `GXSetShapeDash` function, which is described on page 3-139.

Getting and Setting Patterns

QuickDraw GX allows you to specify a pattern to fill a shape when drawn. In particular, you may specify a pattern shape for any line, curve, rectangle, polygon, or path shape that has any framed shape fill or any solid fill. You must always specify pattern shapes in their primitive form.

“The Pattern Structure” on page 3-106 describes the `gxPatternRecord` structure, which you use when retrieving or specifying pattern information. That section also describes what types of shapes you may use as a pattern shape.

You can use the `GXGetStylePattern` function to retrieve the pattern information from a style object and the `GXSetStylePattern` function to specify pattern information for a style object.

The `GXGetShapePattern` and `GXSetShapePattern` functions provide a way to retrieve and specify pattern information for the style object associated with a particular shape.

GXGetStylePattern

You can use the `GXGetStylePattern` function to retrieve the pattern information from a style object.

```
gxPatternRecord *GXGetStylePattern(gxStyle source,
                                   gxPatternRecord *pattern);
```

source The style object whose pattern information you want to retrieve.

pattern A pointer to a `gxPatternRecord` structure. On return, this structure contains the pattern information for the source style object.

function result A copy of the `gxPatternRecord` structure associated with the source style object.

DESCRIPTION

The `GXGetStylePattern` function returns as its function result, and in the `pattern` parameter, a pointer to a `gxPatternRecord` structure containing the pattern information for the style object specified by the `source` parameter.

Geometric Styles

This function creates a new shape to encapsulate the pattern geometry, and places a reference to this shape in the `pattern` field of the returned `gxPatternRecord` structure. You should dispose of this shape when you no longer need it.

Since this function copies the pattern information from the source style, you may make changes to the `gxPatternRecord` structure returned by this function without affecting the source style's pattern information. If you want to change the pattern information in the source style, you must use the `GXSetStylePattern` function.

SPECIAL CONSIDERATIONS

Unless an error results, the `GXGetStylePattern` function creates a shape; you are responsible for disposing of this shape when you no longer need it. See *Inside Macintosh: QuickDraw GX Objects* for information about disposing of QuickDraw GX objects.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`
`parameter_is_nil`

SEE ALSO

For a discussion of patterns, see “Patterns” on page 3-31.

For examples of adding patterns to shapes, see page 3-86 through page 3-91.

For a discussion of the `gxPatternRecord` structure and a description of what types of shapes you can use as pattern shapes, see “The Pattern Structure” on page 3-106.

To specify pattern information for a style object, use the `GXSetStylePattern` function, which is described in the next section.

To retrieve pattern information from a style object associated with a particular shape, use the `GXGetShapePattern` function, which is described on page 3-145. To specify pattern information for a style object associated with a particular shape, use the `GXSetShapePattern` function, which is described on page 3-146.

To determine where pattern shapes are drawn for a particular shape, use the `GXGetShapePatternPositions` function, which is described on page 3-147.

GXSetStylePattern

You can use the `GXSetStylePattern` function to change a style object's pattern information.

```
void GXSetStylePattern(gxStyle target,
                      const gxPatternRecord *pattern);
```

target A reference to the style object whose pattern information you want to change.

pattern A pointer to the new pattern information.

DESCRIPTION

The `GXSetStylePattern` function replaces the pattern information in the style object specified by the `target` parameter with the pattern information provided by the `pattern` parameter. You use the `gxPatternRecord` structure to provide pattern information.

Passing `nil` for the `pattern` parameter indicates that you want no pattern and QuickDraw GX removes any pattern information from the target style.

When you set a style's pattern property using this function, you are effectively changing the pattern for all shapes that share the style.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>style_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>empty_shape_not_allowed</code>	(debugging version)
<code>ignorePlatformShape_not_allowed</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)
<code>nil_style_in_glyph_not_allowed</code>	(debugging version)
<code>complex_glyph_style_not_allowed</code>	(debugging version)
<code>shapeFill_not_allowed</code>	(debugging version)
<code>colorProfile_must_be_nil</code>	(debugging version)

Warnings

`graphic_type_cannot_be_dashed`

Notices (debugging version)

`dash_already_set`
`tags_in_shape_ignored`

SEE ALSO

For a discussion of patterns, see “Patterns” on page 3-31.

For examples of adding patterns to shapes, see page 3-86 through page 3-91.

For a discussion of the `gxPatternRecord` structure and a description of what types of shapes you can use as pattern shapes, see “The Pattern Structure” on page 3-106.

To retrieve pattern information from a style object, use the `GXGetStylePattern` function, which is described on page 3-142.

To retrieve pattern information from a style object associated with a particular shape, use the `GXGetShapePattern` function, which is described in the next section. To specify pattern information for a style object associated with a particular shape, use the `GXSetShapePattern` function, which is described on page 3-146.

To determine where pattern shapes are drawn for a particular shape, use the `GXGetShapePatternPositions` function, which is described on page 3-147.

GXGetShapePattern

You can use the `GXGetShapePattern` function to retrieve the pattern information from the style object associated with a particular shape.

```
gxPatternRecord *GXGetShapePattern(gxShape source,
                                   gxPatternRecord *pattern);
```

source The shape whose pattern information you want to retrieve.

pattern A pointer to a `gxPatternRecord` structure. On return, this structure contains the pattern information for the source shape.

function result A copy of the `gxPatternRecord` structure associated with the source shape’s style object.

DESCRIPTION

The `GXGetShapePattern` function returns as its function result and in the `pattern` parameter a pointer to a `gxPatternRecord` structure containing the pattern information for the style object of the shape specified by the `source` parameter.

This function creates a new shape to encapsulate the pattern geometry, and places a reference to this shape in the `pattern` field of the returned `gxPatternRecord` structure. You should dispose of this shape when you no longer need it.

Since this function copies the pattern information from the source shape’s style, you may make changes to the `gxPatternRecord` structure returned by this function without affecting the source shape’s pattern. If you want to change the pattern information for the source shape, you must use the `GXSetShapePattern` function.

Geometric Styles

SPECIAL CONSIDERATIONS

The `GXGetShapePattern` function may create a shape; you are responsible for disposing of this shape when you no longer need it. See *Inside Macintosh: QuickDraw GX Objects* for information about creating and disposing of objects.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`parameter_is_nil`

SEE ALSO

For a discussion of patterns, see “Patterns” on page 3-31.

For examples of adding patterns to shapes, see page 3-86 through page 3-91.

For a discussion of the `gxPatternRecord` structure and a description of what types of shapes you can use as pattern shapes, see “The Pattern Structure” on page 3-106.

To retrieve pattern information from a style object, use the `GXGetStylePattern` function, which is described on page 3-142. To specify pattern information for a style object, use the `GXSetStylePattern` function, which is described on page 3-144.

To specify pattern information for a style object associated with a particular shape, use the `GXSetShapePattern` function, which is described in the next section.

To determine where pattern shapes are drawn for a particular shape, use the `GXGetShapePatternPositions` function, which is described on page 3-147.

GXSetShapePattern

You can use the `GXSetShapePattern` function to change the pattern information for a style object associated with a particular shape.

```
void GXSetShapePattern(gxShape target,
                      const gxPatternRecord *pattern);
```

<code>target</code>	A reference to the shape whose pattern information you want to change.
<code>pattern</code>	A pointer to the new pattern information.

DESCRIPTION

The `GXSetShapePattern` function replaces the pattern information in the style object of the shape specified by the `target` parameter with the pattern information provided by the `pattern` parameter. You use the `gxPatternRecord` structure to provide pattern information.

Geometric Styles

Passing `nil` for the `pattern` parameter indicates that you want no pattern and QuickDraw GX removes any pattern information from the target shape.

If the `target` shape shares its style object with other shapes, this function makes a copy of the style object, sets the target shape to reference the copy, and changes the `pattern` property of the copy. (However, if the effect of this function would leave the pattern information unchanged, this function does not create a copy of the style object; instead, it returns a notice).

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`parameter_out_of_range`
`pattern_lattice_out_of_range`

Notices (debugging version)

`pattern_already_set`

SEE ALSO

For a discussion of patterns, see “Patterns” on page 3-31.

For examples of adding patterns to shapes, see page 3-86 through page 3-91.

For a discussion of the `gxPatternRecord` structure and a description of what types of shapes you can use as pattern shapes, see “The Pattern Structure” on page 3-106.

To retrieve pattern information from a style object, use the `GXGetStylePattern` function, which is described on page 3-142. To specify pattern information for a style object, use the `GXSetStylePattern` function, which is described on page 3-144.

To retrieve pattern information from a style object associated with a particular shape, use the `GXGetShapePattern` function, which is described on page 3-145.

To determine where pattern shapes are drawn for a particular shape, use the `GXGetShapePatternPositions` function, which is described in the next section.

GXGetShapePatternPositions

You can use the `GXGetShapePatternPositions` function to determine the precise locations where QuickDraw GX draws the shapes that pattern another shape.

```
long GXGetShapePatternPositions(gxShape source,
                               gxPoint positions[]);
```

Geometric Styles

source A reference to the shape whose pattern positions you want to find.

positions An array of pattern positions. On return, this array contains points that indicate the position of the pattern shapes that pattern the source shape.

function result The number of pattern positions returned in the `positions` parameter.

DESCRIPTION

The `GXGetShapePatternPositions` function returns in the `positions` parameter the locations of the pattern shapes as drawn for the source shape.

The function result is the number of pattern positions returned—the number of pattern shapes drawn for the source shape.

If you pass `nil` for the `positions` parameter, the `GXGetShapePatternPositions` function still returns as the function result the number of pattern shapes but it does not return the positions of the pattern shapes.

This function returns 0 if the source shape has no pattern.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

SEE ALSO

For a discussion of patterns, see “Patterns” on page 3-31.

For an example using this function, see “Determining Pattern Positions” on page 3-88.

For a discussion of the `gxPatternRecord` structure and a description of what types of shapes you can use as pattern shapes, see “The Pattern Structure” on page 3-106.

To retrieve pattern information from a style object, use the `GXGetStylePattern` function, which is described on page 3-142. To specify pattern information for a style object, use the `GXSetStylePattern` function, which is described on page 3-144.

To retrieve pattern information from a style object associated with a particular shape, use the `GXGetShapePattern` function, which is described on page 3-145. To specify pattern information for a style object associated with a particular shape, use the `GXSetShapePattern` function, which is described on page 3-146.

Summary of Geometric Styles

Constants and Data Types

Style Attributes

```
enum gxStyleAttributes {
    gxCenterFrameStyle    = 0,          /* center the pen on contour */
    gxSourceGridStyle     = 0x0001,     /* constrain to source grid */
    gxDeviceGridStyle     = 0x0002,     /* constrain to device grid */
    gxInsideFrameStyle    = 0x0004,     /* place pen inside contour */
    gxOutsideFrameStyle   = 0x0008,     /* place pen outside contour */
    gxAutoInsetStyle      = 0x0010     /* don't assume right is in */
};

typedef long gxStyleAttribute;
```

Cap Structure

```
struct gxCapRecord {
    gxCapAttribute attributes; /* modifies behavior of caps */
    gxShape          startCap; /* shape to use at start of contours */
    gxShape          endCap;   /* shape to use at end of contours */
};
```

Cap Attributes

```
enum gxCapAttributes {
    gxLevelStartCap= 0x0001; /* suppress start cap rotation */
    gxLevelEndCap  = 0x0002; /* suppress end cap rotation */
};

typedef long gxCapAttribute;
```

Join Structure

```
struct gxJoinRecord {
    gxJoinAttribute attributes; /* modifies behavior of joins */
    gxShape          join;      /* shape to use at corners */
    Fixed            miter;     /* size limit for sharp joins */
};
```

Geometric Styles

Join Attributes

```
enum gxJoinAttributes {
    gxSharpJoin = 0x0000,    /* use default sharp joins */
    gxCurveJoin = 0x0001,    /* use default curved joins */
    gxLevelJoin = 0x0002     /* suppress join shape rotation */
};

typedef long gxJoinAttribute;
```

Dash Structure

```
struct gxDashRecord {
    gxDashAttribute attributes; /* modifies behavior of dashes */
    gxShape          dash;      /* shape used for dashing */
    Fixed            advance;    /* distance between dashes */
    fract            phase;      /* start offset into the contour */
    Fixed            scale;      /* height of dash (mapped to pen) */
};
```

Dash Attributes

```
typedef enum gxDashAttributes {
    gxBendDash      = 0x0001;    /* distorts shape in 1 dimension */
    gxBreakDash     = 0x0002;    /* places dash contours separately */
    gxClipDash      = 0x0004;    /* clips dashes to pen width */
    gxLevelDash     = 0x0008;    /* suppresses dash rotation */
    gxAutoAdvanceDash = 0x0010;  /* automatically adjusts advances */
};

typedef long gxDashAttribute;
```

Pattern Structure

```
struct gxPatternRecord {
    gxPatternAttribute attributes; /* modifies behavior of pattern */
    gxShape             pattern;   /* shape to use as pattern */
    gxPoint             u;         /* vector for pattern grid */
    gxPoint             v;         /* vector for pattern grid */
};
```

Pattern Attributes

```
enum gxPatternAttributes {
    gxPortAlignPattern = 0x0001, /* align pattern with device */
    gxPortMapPattern   = 0x0002 /* suppress mapping of pattern */
};

typedef long gxPatternAttribute;
```

Functions for Manipulating Geometric Style Properties

Getting and Setting Style Attributes

```
gxStyleAttribute GXGetStyleAttributes
                                   (gxStyle source);
void GXSetStyleAttributes (gxStyle target, gxStyleAttribute attributes);
gxStyleAttribute GXGetShapeStyleAttributes
                                   (gxShape source);
void GXSetShapeStyleAttributes
                                   (gxShape target, gxStyleAttribute attributes);
```

Getting and Setting Curve Error

```
Fixed GXGetStyleCurveError (gxStyle source);
void GXSetStyleCurveError (gxStyle target, Fixed error);
Fixed GXGetShapeCurveError (gxShape source);
void GXSetShapeCurveError (gxShape target, Fixed error);
```

Getting and Setting the Pen Width

```
Fixed GXGetStylePen (gxStyle source);
void GXSetStylePen (gxStyle target, Fixed pen);
Fixed GXGetShapePen (gxShape source);
void GXSetShapePen (gxShape target, Fixed pen);
```

Getting and Setting Caps

```
gxCapRecord *GXGetStyleCap (gxStyle source, gxCapRecord *cap);
void GXSetStyleCap (gxStyle target, const gxCapRecord *cap);
gxCapRecord *GXGetShapeCap (gxShape source, gxCapRecord *cap);
void GXSetShapeCap (gxShape target, const gxCapRecord *cap);
```

Getting and Setting Joins

```

gxJoinRecord *GXGetStyleJoin
                                (gxStyle source, gxJoinRecord *join);
void GXSetStyleJoin             (gxStyle target, const gxJoinRecord *join);
gxJoinRecord *GXGetShapeJoin
                                (gxShape source, gxJoinRecord *join);
void GXSetShapeJoin             (gxShape target, const gxJoinRecord *join);

```

Getting and Setting Dashes

```

gxDashRecord *GXGetStyleDash
                                (gxStyle source, gxDashRecord *dash);
void GXSetStyleDash            (gxStyle target, const gxDashRecord *dash);
gxDashRecord *GXGetShapeDash
                                (gxShape source, gxDashRecord *dash);
void GXSetShapeDash            (gxShape target, const gxDashRecord *dash);
long GXGetShapeDashPositions
                                (gxShape source, gxMapping dashMappings[]);

```

Getting and Setting Patterns

```

gxPatternRecord *GXGetStylePattern
                                (gxStyle source, gxPatternRecord *pattern);
void GXSetStylePattern         (gxStyle target, const gxPatternRecord
                                *pattern);
gxPatternRecord *GXGetShapePattern
                                (gxShape source,
                                gxPatternRecord *pattern);
void GXSetShapePattern         (gxShape target, const gxPatternRecord
                                *pattern);
long GXGetShapePatternPositions
                                (gxShape source, gxPoint positions[]);

```