

Geometric Shapes

This chapter describes the geometric shapes. In particular, it shows you how you can

- define geometries
- create geometric shapes
- manipulate their shape type, shape fill, and geometry properties
- draw the shapes

Before you read this chapter, you should be familiar with some of the information in *Inside Macintosh: QuickDraw GX Objects*. In particular, you should read the chapters “Introduction to QuickDraw GX Objects” and “Shape Objects” in that book.

The next chapter, “Geometric Styles,” discusses the stylistic variations you can apply to geometric shapes.

Chapter 4, “Geometric Operations,” describes the functions QuickDraw GX provides for performing operations on the geometries of geometric shapes—operations such as intersection, union, and so on.

For information about applying colors and transfer modes to geometric shapes, you should read the chapter “Ink Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For information about applying mapping transformations to geometric shapes, clipping geometric shapes, and hit-testing geometric shapes, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

About Geometric Shapes

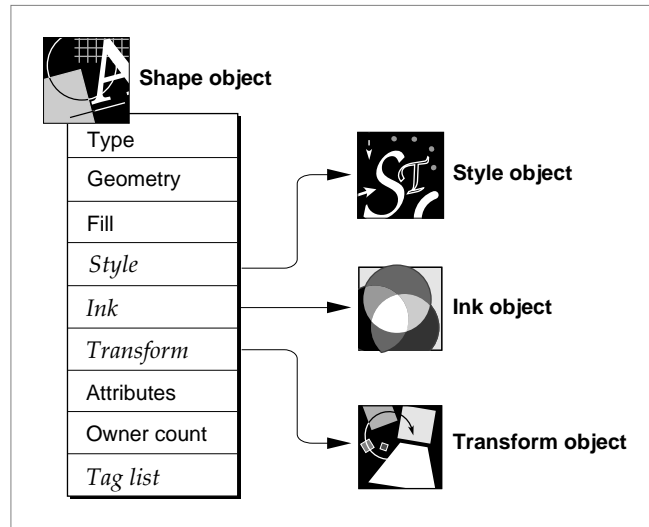
QuickDraw GX represents shapes in memory using a shape object and an associated style, ink, and transform object. QuickDraw GX uses these same objects to represent all types of shapes—graphic as well as typographic.

A shape object has nine **properties**, which are like fields of a data structure with one important exception: you cannot directly examine or change the information stored in a property. Instead, you must use QuickDraw GX functions to examine or alter the value of a property.

Geometric Shapes

Figure 2-1 shows a graphic representation of a shape object and its nine properties.

Figure 2-1 A shape object



The first three properties of a shape object—the shape type, shape geometry, and shape fill—are called the geometric shape properties. These properties are examined in detail in “The Geometric Properties of Shape Objects” beginning on page 2-7. In particular, that section describes how these three properties are used by geometric shapes.

The next three properties of a shape object—the style, ink, and transform properties—are references to the style, ink, and transform objects associated with the shape. Each of these objects contains information that modifies the way QuickDraw GX draws the shape. You can find more information about these objects in *Inside Macintosh: QuickDraw GX Objects*. In addition, you can find specific information about how style objects affect geometric shapes in Chapter 3, “Geometric Styles,” in this book.

The final three properties of a shape object—the shape attributes, the owner count, and the tag list—are the object-related shape properties. You can find information about these properties, and how they affect all types of shapes, in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

QuickDraw GX provides six basic types of geometric shapes and two special types. The six basic types include points, lines, curves, rectangles, polygons, and paths; the two special types include empty shapes and full shapes.

Geometric Shapes

Each of these shape types is examined in detail in “The Geometric Shape Types” beginning on page 2-16. In particular, that section analyzes how each type of geometric shape uses its shape geometry and shape fill, and also discusses the default geometric shapes.

The Geometric Properties of Shape Objects

Every shape object has three geometric properties: the shape type, the shape geometry, and the shape fill. For geometric shapes, these properties define

- the type of shape—for example, a point, a line, or a curve
- the coordinates of the shape—for example, the position where a line starts and ends, or the positions of the corners of a rectangle
- how the shape is filled—for example, whether the shape is framed (drawn as an outline) or solid (drawn as a solid area)

The next three sections examine these properties in more detail.

Shape Type

The **shape type** property of a shape object specifies what type of shape the shape object represents. There are thirteen different QuickDraw GX shape types: one for bitmap shapes, one for picture shapes, three for typographic shapes, and eight for geometric shapes. The eight geometric shape types are:

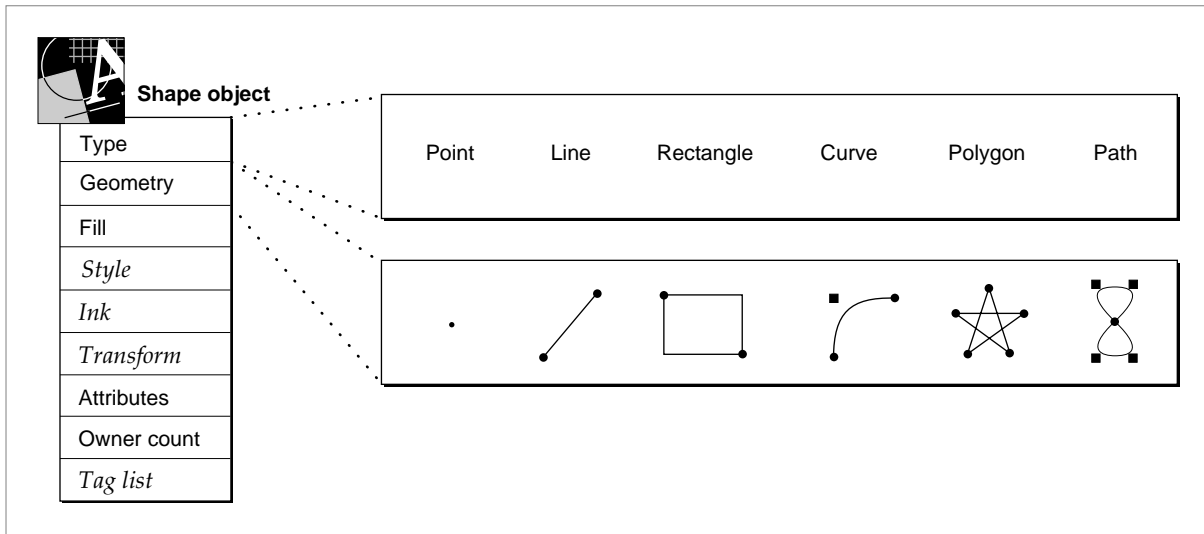
- point
- line
- curve
- rectangle
- polygon
- path
- empty
- full

The value of the shape type property affects the way QuickDraw GX interprets the other properties of the shape. In particular, different types of shapes store substantially different information in their geometry properties. For example, the geometry of a point shape contains only an x-coordinate and a y-coordinate. The geometry of a line contains an x-coordinate and a y-coordinate to define the beginning of the line and an x-coordinate and a y-coordinate to define the end of the line. The geometry of a polygon shape can contain many pairs of (x, y) coordinates.

Geometric Shapes

Figure 2-2 shows a shape object and lists six possible values for its shape type property. This figure also shows a sample geometry for each of the shape types listed. Each geometry is made up of geometric points (specified by (x, y) coordinate pairs) and edges connecting the geometric points. The next section, “Shape Geometry,” discusses geometric points and edges in more detail.

Figure 2-2 The geometric shape types and examples of geometric shape geometries



There are two types of geometric shapes not shown in this figure: the empty shape and the full shape. An empty shape is a shape that has no geometry and covers no area. A full shape is the inverse of an empty shape—it covers all area. You can find more information about these shape types in “Empty Shapes and Full Shapes” beginning on page 2-16.

Shape Geometry

Each type of geometric shape uses the **geometry** property of its shape object in a slightly different manner. For example, empty shapes and full shapes store no information in their geometry, because they require no further geometric information—their shape type says it all.

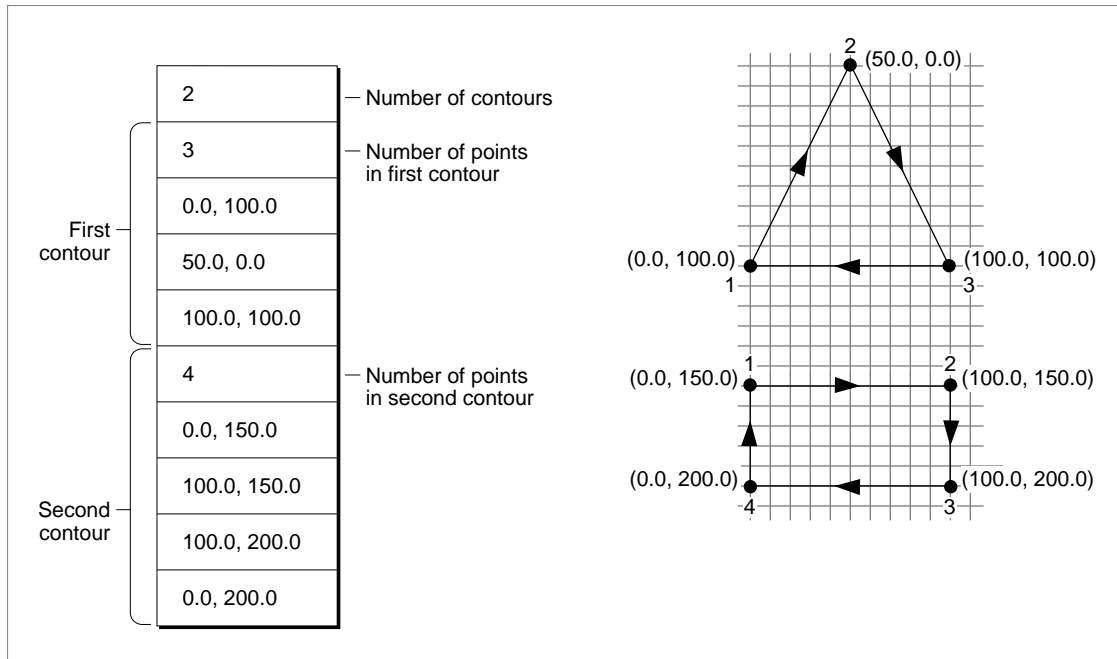
However, for other types of geometric shapes, the shape type does not contain all the geometric information necessary to define the shape. The geometries of these shapes contain (x, y) coordinate pairs called **geometric points**—points that specify the location, dimension, and form of the geometric shapes:

- Point geometries contain one geometric point—an x-coordinate and a y-coordinate—to specify the position of the point shape. See “Point Shapes” on page 2-16 for more information.
- Line geometries contain two geometric points—one point to specify where the line starts and one to specify where the line ends. See “Line Shapes” on page 2-17 for more information.
- Rectangle geometries also contain two geometric points—specifying the positions of opposing corners of the rectangle. See “Rectangle Shapes” on page 2-20 for more information.
- Curve shapes store three geometric points in their geometry—one to specify where the curve starts, another to specify where the curve ends, and another, called the **off-curve control point**, to specify the tangents used to define the curve. See “Curve Shapes” on page 2-18 for more information.
- A polygon shape can contain multiple contours. A **polygon contour** is a series of geometric points connected by straight lines—for example, a V-shape, a triangle, or a hexagon.
- A path geometry can also contain multiple contours, but each **path contour** can contain curves as well as straight lines.

Geometric Shapes

Figure 2-3 shows a polygon shape with a two polygon contours made up of seven geometric points total. This figure shows two views of the polygon geometry: as a list of (x, y) coordinate pairs and as seven geometric points plotted on a geometric grid. This second way of viewing geometries is used frequently throughout this book, as it shows not only the geometric points, but also the implied **edges** that connect them. Typically, the figures in this book do not show the grid, but just the points and edges.

Figure 2-3 A polygon shape with a single contour containing three geometric points



Geometric Shapes

Each geometric point in a geometry has a **geometry index**—if you consider the geometry as a list of geometric points starting from the first geometric point of the first contour to the last geometric point of the last contour, the geometry index of a particular geometric point is its position in this list. For example, in the shape in Figure 2-3, the first point (0.0, 100.0) has a geometry index of 1, the second point (50.0, 0.0) has a geometry index of 2, and the third point (100.0, 100.0) has a geometry index of 3. The first point in the second contour (0.0, 150.0) has a geometry index of 4, as it is the fourth geometric point in the geometry. However, it has a contour index of 1, as it is the first point of its contour. Similarly, the next point (100.0, 150.0) has a geometry index of 5 and a contour index of 2, and so forth.

Notice that each of the three edges of the polygon contour in Figure 2-3 has a direction. The first edge is pointing up and to the right; the second edge is pointing down and to the right; the third edge is pointing to the left. QuickDraw GX takes into consideration the direction that an edge is pointing in a number of circumstances:

- When filling a shape. QuickDraw GX allows you to choose how a shape should be filled. The next section, “Shape Fill,” discusses how the direction of an edge can affect how QuickDraw GX fills a shape.
- When determining the **contour direction** of a contour. In the example in Figure 2-3, both polygon contours have a clockwise contour direction. If their geometric points were reversed, the polygon contours would have a counterclockwise contour direction.
- When determining the inside or outside of a contour. QuickDraw GX normally defines the right side of an edge to be the inside and the left side to be the outside. Since the example in Figure 2-3 has a clockwise contour direction, the inside of the contour corresponds to what you would expect the inside to be. If the contour had a counterclockwise direction, the inside of the contour would correspond to what you might expect the outside to be.

QuickDraw GX uses contour direction and the inside and outside of a shape when applying certain stylistic variations, as described in Chapter 3, “Geometric Styles,” and when performing certain geometric operations, as described in Chapter 4, “Geometric Operations,” of this book.

For more details about the geometries of the various geometric shapes, see “The Geometric Shape Types” beginning on page 2-16.

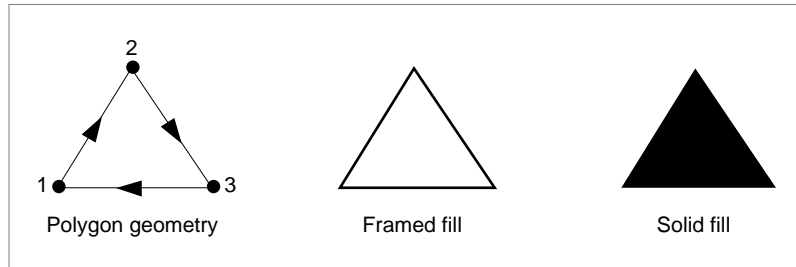
Shape Fill

The shape fill property specifies how QuickDraw GX interprets the geometric points of a geometric shape's geometry during drawing and other operations. There are two basic types of shape fills:

- **Framed fills.** These shape fills indicate that QuickDraw GX should interpret the shape as an outline—as a series of edges.
- **Solid fills.** These shape fills indicate that QuickDraw GX should interpret the shape as a solid area—the edges of the shape represent the boundaries of the area.

Figure 2-4 shows an example of a polygon contour similar to the one in Figure 2-3, and how QuickDraw GX might draw it with a framed fill and with a solid fill.

Figure 2-4 Framed shapes versus solid shapes

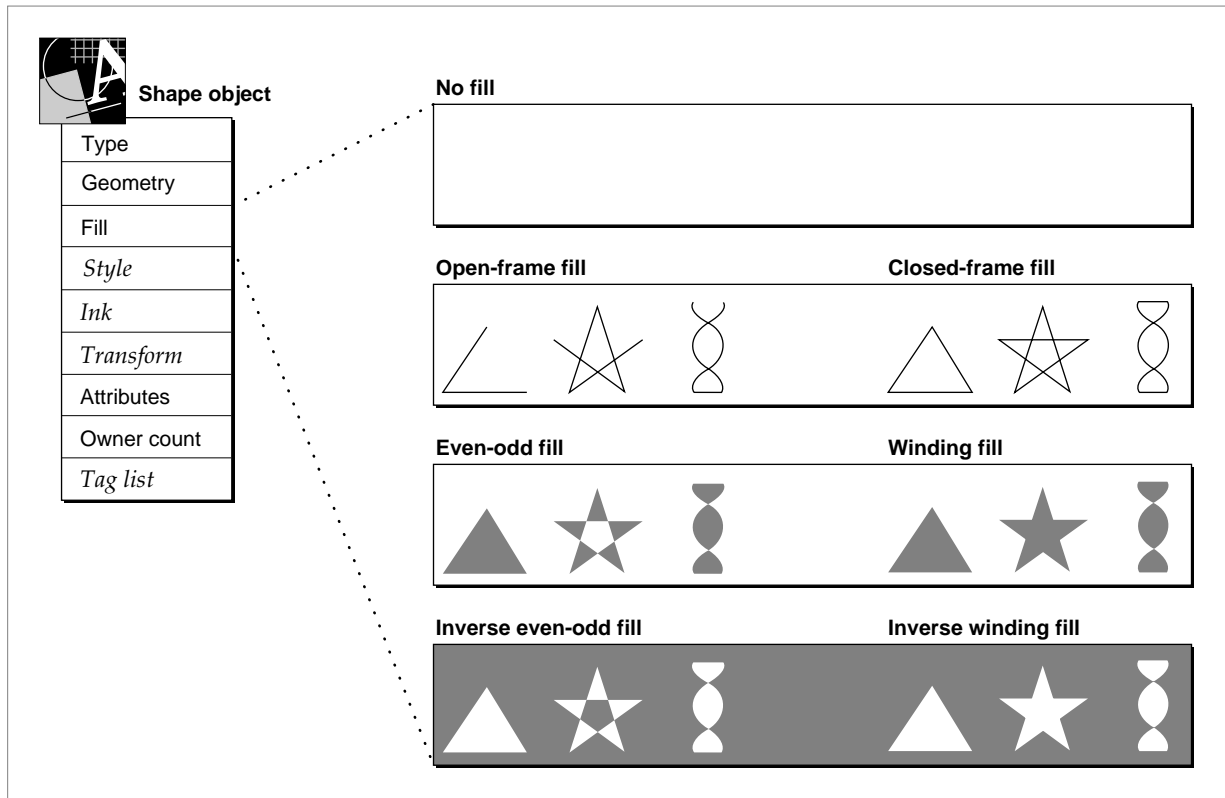


QuickDraw GX actually provides seven types of shape fills:

- no-fill shape fill
- open-frame shape fill (also called *frame fill*)
- closed-frame shape fill (also called *hollow fill*)
- even-odd shape fill (also called *solid fill*)
- winding shape fill
- inverse even-odd shape fill (also called *inverse fill* and *inverse solid fill*)
- inverse winding shape fill

Figure 2-5 shows these shape fills and the effect they have on three sample geometries.

Figure 2-5 The various shape fills and examples of their effects



The no-fill shape fill specifies that QuickDraw GX should not draw the shape. You can use this shape fill to hide a shape. You can specify the no-fill shape fill for any shape type.

The open-frame shape fill specifies that QuickDraw GX should draw a shape as a connected set of edges. The closed-frame shape fill indicates that QuickDraw GX should also connect the last geometric point of a contour to the first geometric point of that contour.

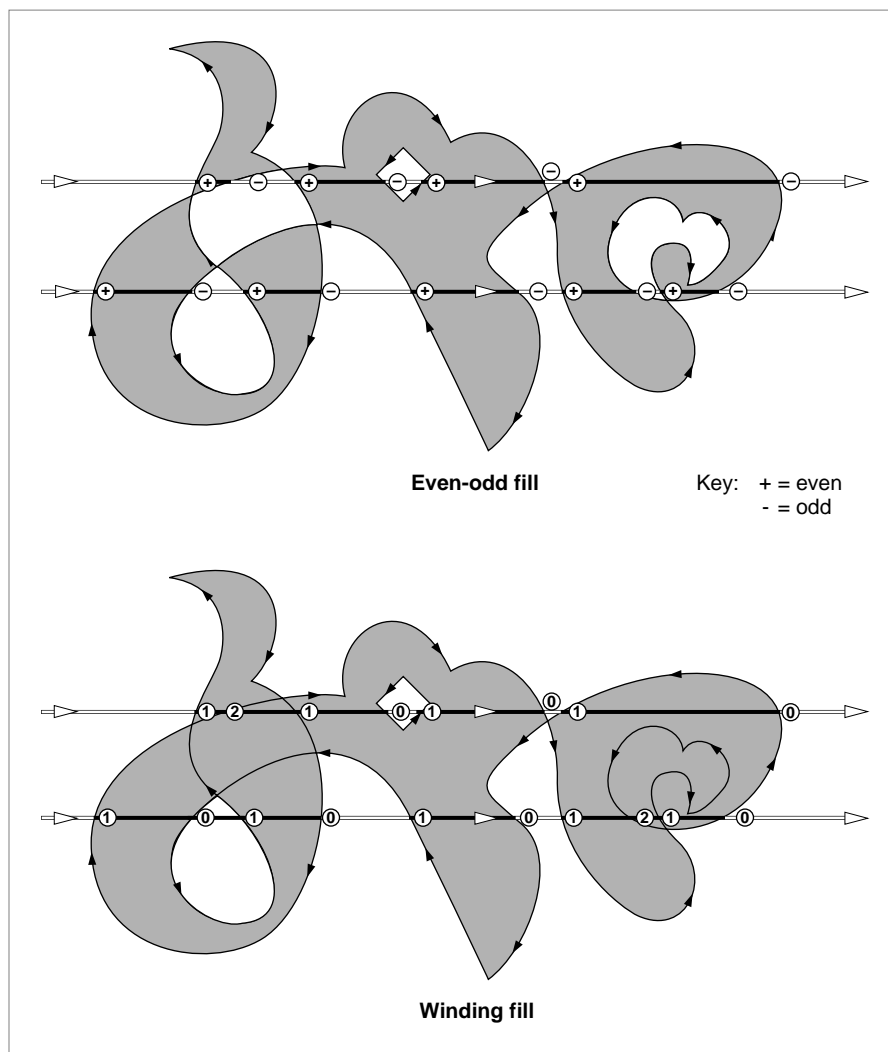
The even-odd shape fill and the winding shape fill indicate that QuickDraw GX should interpret the shape as a solid area—the edges of the shape represent the boundaries of the area. These two shape fills differ in the algorithm they use to determine what area to include in the shape.

The even-odd shape fill indicates that QuickDraw GX should use the **even-odd rule** to determine what area lies inside a shape. As QuickDraw GX scans a shape horizontally, it fills the area between every other pair of edges, as shown in Figure 2-6.

Geometric Shapes

The winding shape fill indicates that QuickDraw GX should use the **winding-number rule** to determine what area lies inside a shape. As QuickDraw GX scans a shape horizontally, it increments a counter the first time it crosses an edge of the shape. It also notices whether the contour was directed up or down at that edge. As QuickDraw GX continues to scan the shape horizontally, everytime it crosses another edge pointed in the same direction (up or down), it increments the counter, and when it crosses an edge pointing in the opposite direction (down or up), it decrements the counter. Wherever along the horizontal scan line the counter is not zero, QuickDraw GX fills the area, as is shown in Figure 2-6.

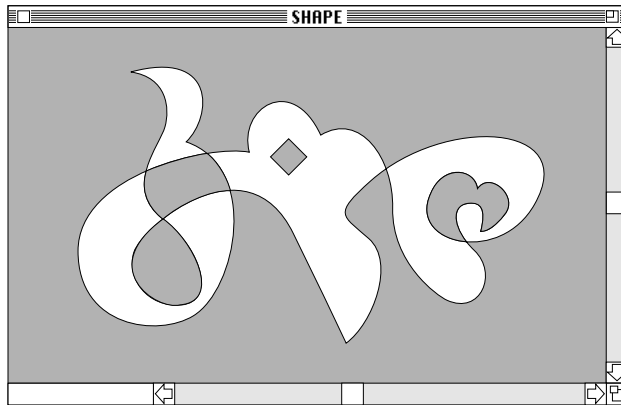
Figure 2-6 The even-odd rule and winding-number rule algorithms



Geometric Shapes

The inverse even-odd shape fill indicates the inverse of the even-odd shape fill, as shown in Figure 2-7.

Figure 2-7 The inverse even-odd shape fill



Similarly, the inverse winding shape fill indicates the inverse of the winding shape fill.

Not all shape fills are appropriate for all types of geometric shapes. For example, a rectangle shape can have a closed-frame shape fill but not an open-frame shape fill; a line shape can only have a no-fill or an open-frame shape fill.

See the sections on each shape type, beginning on page 2-16, for a complete discussion of the shape fills that are allowed for each shape type.

The shape fill does more than affect the way a shape is drawn; it affects the fundamental behavior of a shape. Two shapes with the same geometry that have different shape fills can exhibit vastly different geometric behaviors. For example, the shape fill can affect

- stylistic variations, which are described in Chapter 3, “Geometric Styles,” in this book
- shape measurements and other geometric operations, which are discussed in Chapter 4, “Geometric Operations,” in this book (As an example, a polygon with the closed-frame shape fill might simplify to a rectangle. However, the same polygon with the open-frame fill might not simplify at all.)
- hit-testing, which is described in the chapter “Transform Objects” and the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*

For examples of how shape fill affects the behavior of shapes, see

- “Polygon Shapes” beginning on page 2-22
- “Path Shapes” beginning on page 2-25
- “Creating and Drawing Polygons” beginning on page 2-45
- “Creating and Drawing Paths” beginning on page 2-55

The Geometric Shape Types

QuickDraw GX provides eight types of geometric shapes: empty shapes, full shapes, point shapes, line shapes, curve shapes, rectangle shapes, polygon shapes, and path shapes.

The following sections examine each of these shape types in detail. In particular, these sections discuss how the different types of shapes use their geometry and shape fill properties, and what the default values are for properties of each type of shape.

Empty Shapes and Full Shapes

Empty shapes and full shapes are the only geometric shapes with no information stored in the geometry property.

An **empty shape** is a shape with no geometry. When you draw an empty shape, nothing appears. You can use an empty shape when creating other types of shapes. For example, you can create an empty shape and then build it into a polygon shape, adding one contour at a time.

A **full shape** is a shape that covers the largest area possible. When you draw a full shape, QuickDraw GX fills in the entire drawable area of the full shape's view port (paying attention to the clipping information stored in the full shape's transform). You can use a full shape when erasing an area.

Point Shapes

The **point shape** is the simplest of the geometric shapes. Its geometry consists of a single geometric point—a single (x, y) coordinate pair.

Point shapes must always have the open-frame shape fill or the no-fill shape fill.

A point shape's style determines how QuickDraw GX draws the point. If a point's style has a pen width of 0, which is the default pen width, QuickDraw GX draws the point as a single pixel on the output device. If the style has a pen width greater than 0, QuickDraw GX draws the point only if the style also has a start cap. The next chapter, "Geometric Styles," discusses these aspects of the style object in more detail.

When you create a new point shape, QuickDraw GX makes a copy of the default point shape. The default point shape has these properties:

- owner count: 1
- tag list: no tags
- shape attributes: no attributes
- shape type: point type
- shape fill: open-frame fill
- geometry: (0.0, 0.0)

Geometric Shapes

You may change the properties of the default point shape, which effectively changes the behavior of the functions that create point shapes. However, when creating a new point shape, QuickDraw GX always initializes the owner count to 1 and the geometry to (0.0, 0.0), even if you have specified other values for the default point shape.

For examples of creating and drawing point shapes, see “Creating and Drawing Points” beginning on page 2-29.

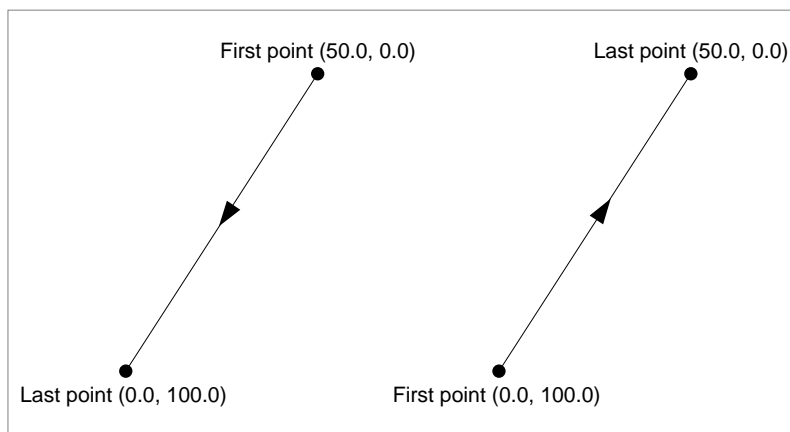
Line Shapes

The geometry of a **line shape** consists of two geometric points: a first point and a last point. Because the points are ordered, a line points in a certain direction.

Line shapes must always have the open-frame shape fill or the no-fill shape fill.

Figure 2-8 shows two line shapes. The geometries of these two lines have the same geometric points, but in the opposite order. Therefore, the two lines point in opposite directions.

Figure 2-8 Two lines



If a line shape uses the default style information, the direction of the line does not affect how QuickDraw GX draws the line. However, when you add stylistic variations (such as pen width, pen placement, and dashes) to a line shape, the direction of the line can affect how QuickDraw GX draws the line. See the next chapter, “Geometric Styles,” for information about how you can add stylistic variations to a line.

When you create a new line shape, QuickDraw GX makes a copy of the default line shape. The default line shape has these properties:

- owner count: 1
- tag list: no tags
- shape attributes: no attributes

Geometric Shapes

- shape type: line type
- shape fill: open-frame fill
- geometry: (0.0, 0.0), (0.0, 0.0)

You may change the properties of the default line shape, which effectively changes the behavior of the functions that create line shapes. However, when creating a new line shape, QuickDraw GX always initializes the owner count to 1 and the geometry to (0.0, 0.0), (0.0, 0.0), even if you have specified other values for the default line shape.

For examples of creating and drawing line shapes without stylistic variations, see “Creating and Drawing Lines” beginning on page 2-36.

For examples of creating and drawing lines with stylistic variations, see the next chapter, “Geometric Styles.”

Curve Shapes

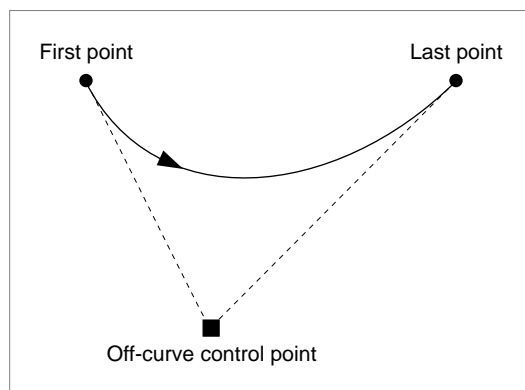
The geometry of a **curve shape** consists of three geometric points: a first point, a last point, and an off-curve control point that determines the tangents of the curve. The curve described by these three points is a quadratic Bézier curve—the same type of curve used to describe TrueType fonts.

Because a curve’s geometric points are ordered, a curve has direction. As with line shapes, direction affects the drawing of a curve only after you apply stylistic variations, which are discussed in the next chapter, “Geometric Styles.”

Curve shapes must always have the open-frame shape fill or no fill shape fill.

Figure 2-9 shows an example of a curve shape. In this example, the first point is (50.0, 50.0), the last point is (200.0, 50.0) and the off-curve control point is (100.0, 150.0).

Figure 2-9 A quadratic Bézier curve

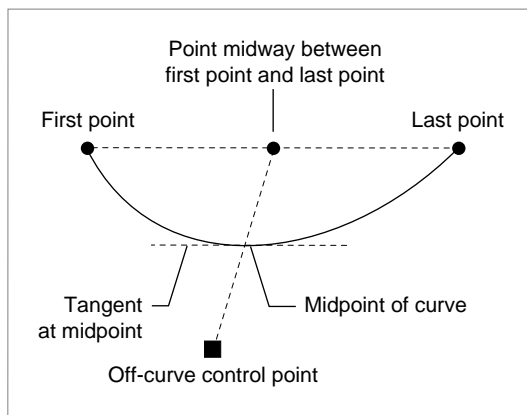


Geometric Shapes

Quadratic Bézier curves have the following characteristics:

- A line connecting the first point and the off-curve control point describes the tangent of the curve at the first point.
- A line connecting the off-curve control point and the last point describes the tangent of the curve at the last point.
- The curve is always contained by the triangle formed by the three geometric points.
- The midpoint of the curve is halfway between the off-curve control point and the point midway between the first point and last point, as shown in Figure 2-10.

Figure 2-10 Finding the midpoint of a curve



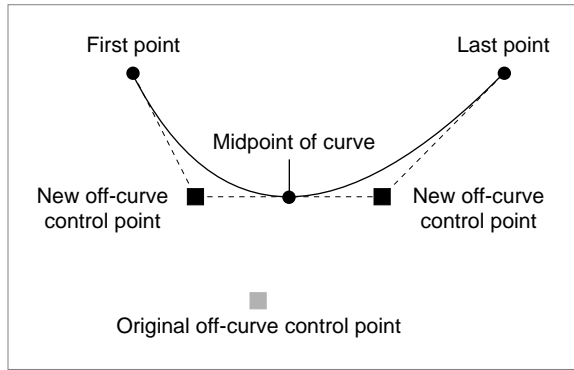
You can divide a quadratic Bézier curve into two smaller quadratic Bézier curves:

- One smaller curve extends from the first point to the midpoint of the original curve. The new off-curve control point is the point midway between the first point and the original off-curve control point.
- The other smaller curve extends from the midpoint to the last point of the original curve. The new off-curve control point is the point midway between the original off-curve control point and the last point.

Geometric Shapes

Figure 2-11 shows a curve divided into two smaller curves.

Figure 2-11 Dividing a curve into two smaller curves



When you create a new curve shape, QuickDraw GX makes a copy of the default curve shape. The default curve shape has these properties:

- owner count: 1
- tag list: no tags
- shape attributes: no attributes
- shape type: curve type
- shape fill: open-frame fill
- geometry: (0.0, 0.0), (0.0, 0.0), (0.0, 0.0)

You may change the properties of the default curve shape, which effectively changes the behavior of the functions that create curve shapes. However, when creating a new curve shape, QuickDraw GX always initializes the owner count to 1 and the geometry to (0.0, 0.0), (0.0, 0.0), (0.0, 0.0), even if you have specified other values for the default curve shape.

For examples of creating and drawing curve shapes without stylistic variations, see “Creating and Drawing Curves” beginning on page 2-41.

For examples of creating and drawing curves with stylistic variations, see the next chapter, “Geometric Styles.”

Rectangle Shapes

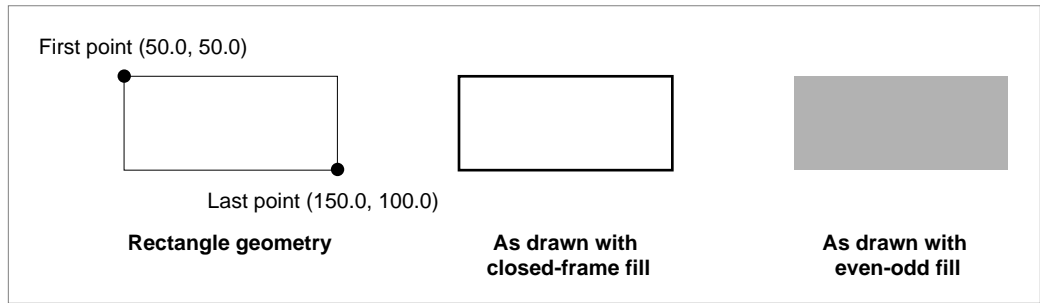
The geometry of a **rectangle shape** consists of two geometric points. Typically, these geometric points represent the upper-left and lower-right corners of the rectangle; however, you can specify any corner as the first geometric point and the diagonally opposite corner as the second geometric point.

Geometric Shapes

Rectangle shapes can have any shape fill except the open-frame shape fill.

Figure 2-12 shows a rectangle geometry and how that rectangle is drawn with a closed-frame shape fill and how it is drawn with an even-odd shape fill.

Figure 2-12 A rectangle geometry shown framed and filled

**Note**

Although you may specify a rectangle's geometric points in any order, QuickDraw GX functions that calculate rectangles always return rectangles with the upper-left corner as the first geometric point and the lower-right corner as the second geometric point. ♦

When you create a new rectangle shape, QuickDraw GX makes a copy of the default rectangle shape. The default rectangle shape has these properties:

- owner count: 1
- tag list: no tags
- shape attributes: no attributes
- shape type: rectangle type
- shape fill: even-odd fill
- geometry: (0.0, 0.0), (0.0, 0.0)

You may change the properties of the default rectangle shape, which effectively changes the behavior of the functions that create rectangle shapes. However, when creating a new rectangle shape, QuickDraw GX always initializes the owner count to 1 and the geometry to (0.0, 0.0), (0.0, 0.0), even if you have specified other values for the default rectangle shape.

For examples of creating and drawing rectangle shapes without stylistic variations, see "Creating and Drawing Rectangles" beginning on page 2-43.

For examples of creating and drawing rectangles with stylistic variations, see the next chapter, "Geometric Styles."

Geometric Shapes

Polygon Shapes

A **polygon contour** is a series of geometric points connected by straight lines. A **polygon shape** may include any number of polygon contours.

Implementation Note

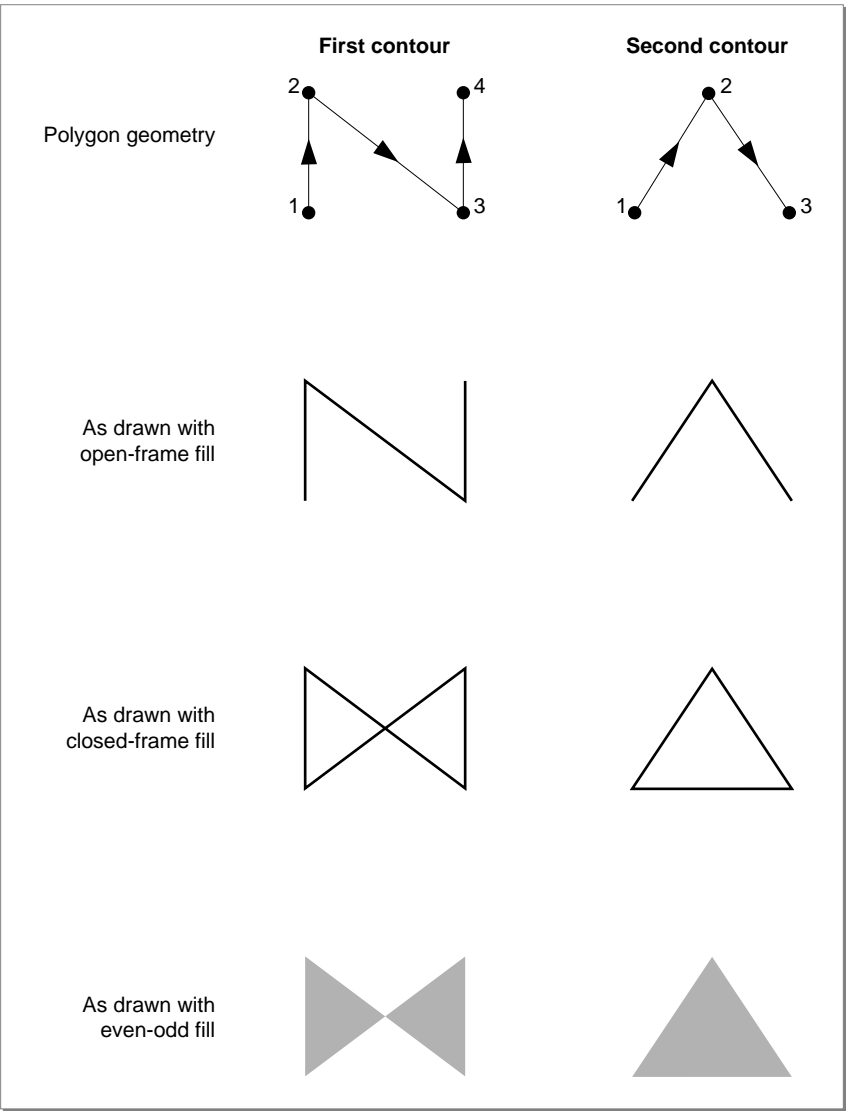
In version 1.0 of QuickDraw GX, a single polygon contour can have between 1 and 32,767 geometric points. The geometry of a polygon shape can have between 0 and 32,767 polygon contours. The total size of a polygon geometry may not exceed 2,147,483,647 bytes. ♦

Polygon shapes may have any shape fill.

Figure 2-13 shows a polygon shape that contains two separate contours. The shape is shown four times:

- as a polygon shape geometry
- as drawn with the open-frame shape fill
- as drawn with the closed-frame shape fill
- as drawn with even-odd shape fill

Figure 2-13 A polygon shape with two polygon contours



The first contour in Figure 2-13 has four geometric points and the second contour has three geometric points.

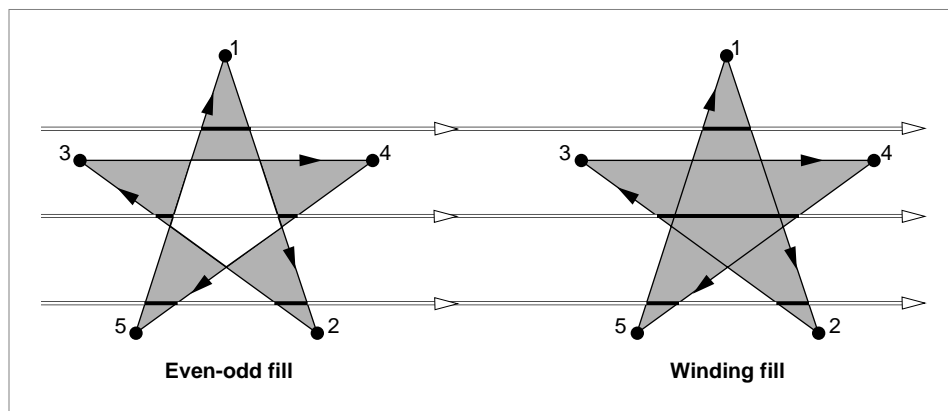
Geometric Shapes

The index of a geometric point within its contour is called its **contour index**. The geometric points in the first contour in Figure 2-13 have contour indexes ranging from 1 to 4, and the geometric points in the second contour in Figure 2-13 have contour indexes ranging from 1 to 3. These contour indexes are shown in the top part of the figure.

Since contours and geometric points are ordered, each geometric point can be numbered from the first geometric point of the first contour to the last geometric point of the last contour. This number is called a geometric point's **geometry index**. Since the polygon geometry in Figure 2-13 has seven geometric points total, these points have geometry indexes ranging from 1 to 7. You use geometry indexes and contour indexes of geometric points when editing polygon geometries. For examples, see "Editing Polygon Parts" beginning on page 2-82.

If the contours of a polygon shape cross over one another, or if a polygon shape contains contours that lie within other contours, the even-odd shape fill and the winding shape fill may fill the polygon shape differently, as shown in Figure 2-14.

Figure 2-14 A polygon drawn with the even-odd and winding shape fills



When you create a new polygon shape, QuickDraw GX makes a copy of the default polygon shape. The default polygon shape has these properties:

- owner count: 1
- tag list: no tags
- shape attributes: no attributes
- shape type: polygon type
- shape fill: even-odd fill
- geometry: 0 contours, 0 points

Geometric Shapes

You may change the properties of the default polygon shape, which effectively changes the behavior of the functions that create polygon shapes. However, when creating a new polygon shape, QuickDraw GX always initializes the owner count to 1 and the geometry to 0 contours with 0 points, even if you have specified other values for the default polygon shape.

For examples of creating and drawing polygon shapes without stylistic variations, see “Creating and Drawing Polygons” beginning on page 2-45.

For examples of creating and drawing polygons with stylistic variations, see the next chapter, “Geometric Styles.”

Path Shapes

A **path contour**, like a polygon contour, is defined by a series of geometric points. However, a path contour can contain off-curve control points as well as on-curve points; therefore, a path contour can contain curves as well as straight lines. A **path shape** may include any number of path contours.

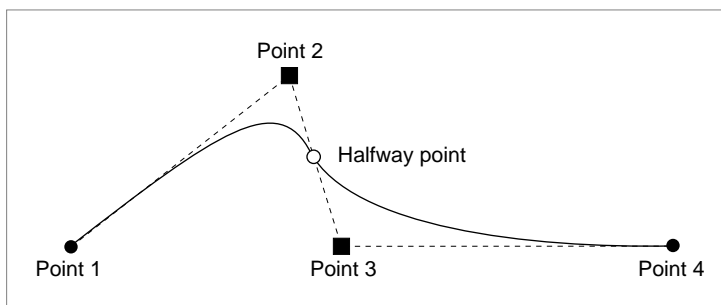
Implementation Note

In version 1.0 of QuickDraw GX, a single path contour can have between 0 and 32,767 geometric points. The geometry of a path shape can between 0 and 32,767 polygon contours. The total size of a path geometry may not exceed 2,147,483,647 bytes. ♦

Every path contains an array of **control bits** that specify which geometric points are on curve and which geometric points are off curve. QuickDraw GX connects two consecutive on-curve points with a straight line. If two on-curve points have an off-curve point between them, QuickDraw GX connects the two on-curve points with a quadratic Bézier curve, using the geometric point between them as the off-curve control point.

QuickDraw GX allows a path to have two or more consecutive off-curve control points. In this case, each pair of consecutive off-curve points implies an on-curve point midway between them, as represented by the small hollow circle in Figure 2-15.

Figure 2-15 A path with two consecutive off-curve points

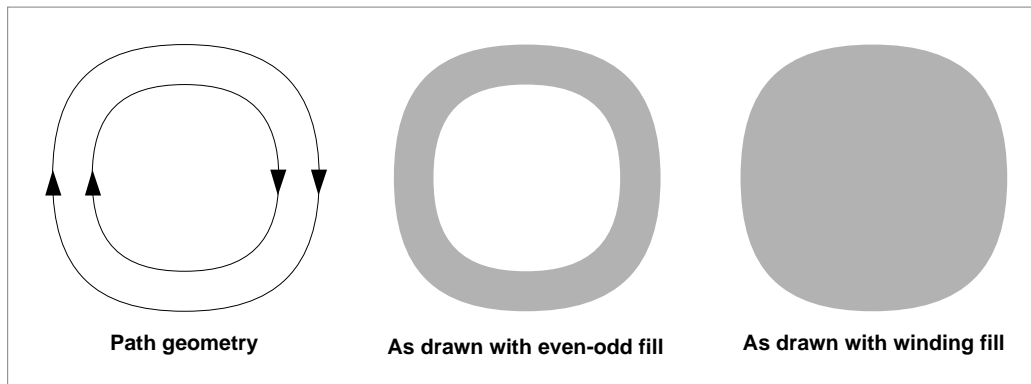


Geometric Shapes

Path shapes may have any shape fill—including open-frame shape fill. However, a path may not have the open-frame shape fill if the first point or the last point of any path contour is an off-curve point.

If the contours of a path shape cross over one another, or if a path shape contains contours that lie within other contours, the even-odd shape fill and the winding shape fill may fill the path shape differently, as shown in Figure 2-16.

Figure 2-16 A path shape filled with the even-odd and winding shape fills



Contour direction affects how QuickDraw GX fills a path when the path has the winding shape fill. In the example in Figure 2-16, if the inner contour has the opposite contour direction from the outer contour, the winding shape fill works in the same manner as the even-odd shape fill. For more information, see the next section, “Shape Fill.” For examples, see “Creating and Drawing Paths” beginning on page 2-55.

When you create a new path shape, QuickDraw GX makes a copy of the default path shape. The default path shape has these properties:

- owner count: 1
- tag list: no tags
- shape attributes: no attributes
- shape type: path type
- shape fill: even-odd fill
- geometry: 0 contours, 0 points

Geometric Shapes

You may change the properties of the default path shape, which effectively changes the behavior of the functions that create path shapes. However, when creating a new path shape, QuickDraw GX always initializes the owner count to 1 and the geometry to 0 contours with 0 points, even if you have specified other values for the default paths shape.

For examples of creating and drawing path shapes without stylistic variations, see “Creating and Drawing Paths” beginning on page 2-55.

For examples of creating and drawing paths with stylistic variations, see the next chapter, “Geometric Styles.”

Using Geometric Shapes

This section shows you how to create, edit, and draw geometric shapes. In particular, this section shows you how to

- create and draw empty and full shapes
- create point, line, curve, rectangle, polygon, and path shapes
- draw points, lines, curves, rectangles, polygons, and paths
- create framed and solid shapes
- convert a shape from one shape type to another
- replace the geometry of a shape
- replace geometric points within a shape’s geometry
- insert geometric points into and remove geometric points from a shape’s geometry

All of the sample functions in this section create geometric shapes with default style, ink, and transform information. All shapes are black; framed shapes have one-pixel-wide contours; and the shapes are not rotated, skewed, and so on. For examples of the many stylistic variations you can apply to geometric shapes, see Chapter 3 of this book, “Geometric Styles.” For information about inks and transforms, see *Inside Macintosh: QuickDraw GX Objects*.

Many of the sample functions in this section create geometric shapes and, to do so, they specify geometric points for the shapes’ geometries. Since a geometric point contains two fixed-point values (of type `Fixed`), the sample functions in this section must convert integer constants to fixed-point constants when specifying a geometric point. QuickDraw GX provides the `GXIntToFixed` macro, which performs this conversion by shifting the integer value 16 bits to the left:

```
#define GXIntToFixed(a) ((Fixed)(a) << 16)
```

Geometric Shapes

QuickDraw GX also provides the `ff` macro as a convenient alias:

```
#define ff(a) GXIntToFixed(a)
```

A few of the sample functions in this section specify fractional values for geometric point coordinates. To convert a floating-point value (of type `float`) to a fixed-point value (type `Fixed`), QuickDraw GX provides the `GXFloatToFixed` macro:

```
#define GXFloatToFixed(a) ((Fixed)((float)(a) * fixed1))
```

and the synonymous `fl` macro:

```
#define fl(a) GXFloatToFixed(a)
```

IMPORTANT

The `GXIntToFixed` macro has substantially faster performance than the `GXFloatToFixed` macro. Whenever possible, you should choose the `GXIntToFixed` macro over the `GXFloatToFixed` macro. ▲

Creating and Drawing Empty Shapes and Full Shapes

To create an empty shape or a full shape, you use the function `GXNewShape`, which is described in full in the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

To create an empty shape, you could define a shape reference and then call the `GXNewShape` function:

```
gxShape anEmptyShape;
```

```
anEmptyShape = GXNewShape(gxEmptyType);
```

Although you can draw this shape with the `GXDrawShape` function, nothing will appear. However, you can use empty shapes for other purposes. For example, you can create an empty shape and then add geometric points to it using the `SetShapeParts` function, building other types of shapes as you add points. See “Editing Shape Parts” beginning on page 2-93 for examples of this function.

Geometric Shapes

To create a full shape, you can use this code:

```
gxShape aFullShape;

aFullShape = GXNewShape(gxFullType);
```

You can then draw the full shape to cover the entire area of the shape's view ports. For example, you could use the full shape to erase an area, or you could set the color of the full shape and draw it to create a colored background before drawing other shapes.

Creating and Drawing Points

QuickDraw GX provides a number of methods to create and draw geometric shapes. In general, to draw a shape you must first define a geometry. You can then draw the shape in one of two ways:

- You can draw the geometry directly—without having to create a shape object.
- You can create a shape object to encapsulate the geometry and then draw the shape.

The first sample function in this section, shown in Listing 2-1, uses the first method—it draws a point without creating a point shape.

To draw the point, this sample function first defines a point geometry, which is represented by a point structure (of type `gxPoint`):

```
struct gxPoint {
    Fixed    x;
    Fixed    y;
};
```

The value in the `x` field specifies horizontal distance from the origin; greater values indicate distances further to the right. The value in the `y` field specifies vertical distance from the origin; greater values indicate distances further down.

Note

The coordinates of a shape's geometry go through a number of transformations before the shape is actually drawn. Where the shape is drawn depends not only on the values of the shape's geometry, but also on the shape's associated transform and view port objects. If you use the default transform and view port information, the coordinates in a shape's geometry represent units of 1/72 inch and the origin is the upper-left corner of the view port. See *Inside Macintosh: QuickDraw GX Objects* for more information about the coordinate systems of QuickDraw GX. ♦

Geometric Shapes

Since each coordinate of a point must be a fixed-point value, the sample function in Listing 2-1 uses the `GXIntToFixed` macro to convert integer constants to fixed-point constants.

The sample function then draws the point using the `GXDrawPoint` function. The `GXDrawPoint` function takes a pointer to a `gxPoint` structure as its only parameter and draws the corresponding point. When drawing the point, it uses the style, ink, and transform information associated with the default point shape.

Listing 2-1 Drawing a point without creating a point shape

```
void DrawASinglePoint(void)
{
    static gxPoint aPointGeometry = {GXIntToFixed(5),
                                     GXIntToFixed(5)};

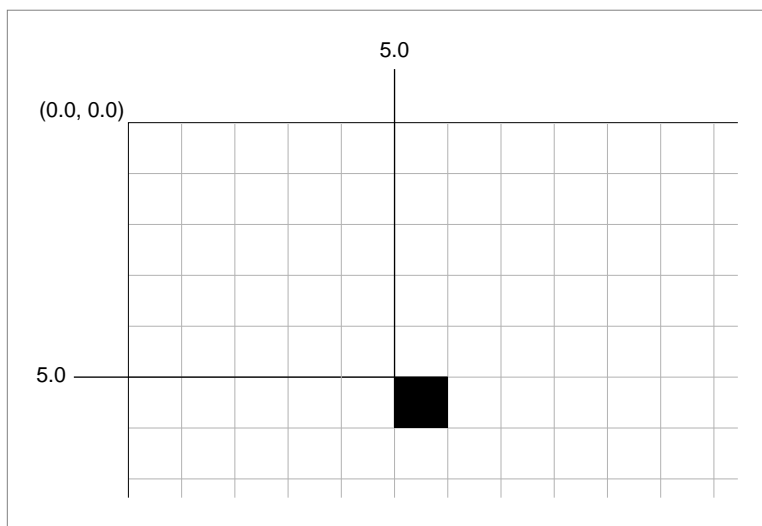
    GXDrawPoint(&aPointGeometry);
}
```

QuickDraw GX provides the `ff` macro as an alias for the `GXIntToFixed` macro. In the example in Listing 2-1, the point coordinates could be specified with this line of code:

```
static gxPoint aPointGeometry = {ff(5), ff(5)};
```

The rest of the examples in “Using Geometric Shapes” use this convenient alternative.

Figure 2-17 shows the result of the sample function from Listing 2-1.

Figure 2-17 A point


Listing 2-1 defines the point at location (5.0, 5.0), which lies at the intersection of two infinitely thin grid lines, and therefore is infinitely thin itself. However, when QuickDraw GX draws this point shape, it draws it as a single pixel—the pixel lying down and to the right of the point itself, as shown in Figure 2-17. QuickDraw GX only draws this single-pixel type of point, called a **hairline point**, if the pen width property of the style object associated with the point shape has a value of 0, which is the default value for this property. If the pen width is greater than 0.0, QuickDraw GX does not draw the point, unless it has a start cap, in which case only the start cap is drawn. For more information about the pen width property and cap property of style objects and how they affects the drawing of point shapes, see the chapter “Geometric Styles,” in this book.

Although you may sometimes want to draw a shape without creating a shape object for it, you will frequently want to create a shape object before drawing a shape. Creating a shape object has many advantages; for example, it allows you to provide custom style, ink, and transform information before drawing the shape.

QuickDraw GX provides three main methods for creating geometric shapes:

- You can call a type-specific function, such as `GXNewPoint`, which requires you to provide a pointer to the shape’s desired geometric structure.
- You can call the `GXNewShapeVector` function, which requires you to specify the shape type and provide a pointer to the shape’s desired geometric structure.
- You can call the `GXNewShape` function, which requires you to specify the desired shape type, and then call a type-specific function, such as `GXSetPoint`, to set the geometry.

The sample functions in Listing 2-2, Listing 2-3, and Listing 2-4 show how to create a point shape using these three methods.

Listing 2-2 uses the `GXNewPoint` function to create a point shape given a pointer to a point geometry.

Listing 2-2 Creating a point shape with the `GXNewPoint` function

```
void CreatePointShape(void)
{
    gxShape      aPointShape;
    static gxPoint aPointGeometry = {ff(5), ff(5)};

    aPointShape = GXNewPoint(&aPointGeometry);

    GXDrawShape(aPointShape);

    GXDisposeShape(aPointShape);
}
```

Geometric Shapes

Listing 2-3 uses the `GXNewShapeVector` function to create a point shape. The `GXNewShapeVector` function requires two parameters:

- the shape type of the shape you want to create
- an array of fixed-point values that represent the shape's geometry

In this example, the desired shape type is `gxPointType` and the geometry is specified as an array of two fixed-point values representing the coordinates of the point's geometry. When using the `GXNewShapeVector` function to create shapes more complicated than point shapes, you need to provide more values in this array.

Listing 2-3 Creating a point shape with the `GXNewShapeVector` function

```
void CreatePointShape(void)
{
    gxShape      aPointShape;
    static Fixed aPointGeometry[] = {ff(5), ff(5)};

    aPointShape = GXNewShapeVector(gxPointType, aPointGeometry);

    GXDrawShape(aPointShape);

    GXDisposeShape(aPointShape);
}
```

Listing 2-4 creates a point shape using the `GXNewShape` function. The `GXNewShape` function requires only that you specify the type of shape to create. You do not have to specify any values for the geometric points of the shape's geometry—the `GXNewShape` function initializes the point geometry to (0.0, 0.0).

Geometric Shapes

To set the values of the point shape's geometry once it's created, the sample function in Listing 2-4 uses the `GXSetPoint` function. This function takes a reference to the shape and a pointer to the desired geometry as its parameters.

Listing 2-4 Creating a point shape with the `GXNewShape` and `GXSetPoint` functions

```
void CreatePointShape(void)
{
    gxShape  aPointShape;

    static gxPoint aPointGeometry = {ff(5), ff(5)};

    aPointShape = GXNewShape(gxPointType);
    GXSetPoint(aPointShape, &aPointGeometry);

    GXDrawShape(aPointShape);

    GXDisposeShape(aPointShape);
}
```

The sample functions in Listing 2-2, Listing 2-3, and Listing 2-4 all use the `GXDrawShape` function to draw the point after the point shape has been created. The resulting point is the same for all three examples; it appears as shown in Figure 2-17.

Geometric Shapes

You can use the `GXSetPoint` function to replace a point shape's geometry any number of times. The sample function in Listing 2-5 creates a point shape, sets its geometry using the `GXSetPoint` function, draws the point, replaces its geometry using the `GXSetPoint` function, and draws the point again.

Listing 2-5 Using the `GXSetPoint` function to replace a point shape's geometry

```
void ReplacePointShapeGeometry(void)
{
    gxShape  aPointShape;

    static gxPoint aPointGeometry = {ff(5), ff(5)};
    static gxPoint anotherPointGeometry = {ff(13), ff(8)};

    aPointShape = GXNewShape(gxPointType);
    GXSetPoint(aPointShape, &aPointGeometry);
    GXDrawShape(aPointShape);

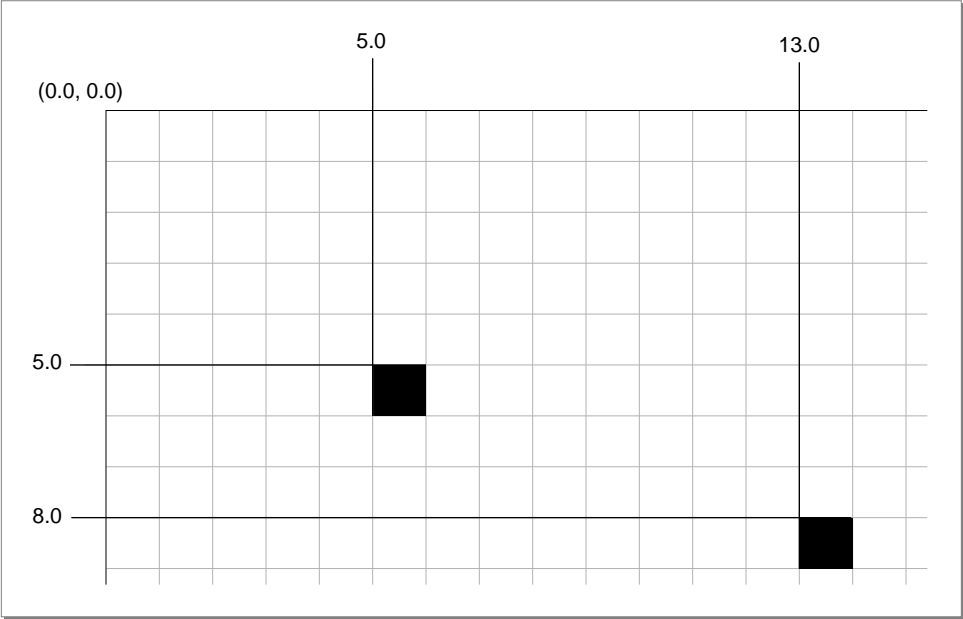
    GXSetPoint(aPointShape, &anotherPointGeometry);
    GXDrawShape(aPointShape);

    GXDisposeShape(aPointShape);
}
```

Geometric Shapes

Figure 2-18 depicts the results of this sample function.

Figure 2-18 Two different point geometries



Geometric Shapes

Most of the sample functions discussed in “Using Geometric Shapes” create shape objects. If you create a shape object using any of the methods discussed, you are responsible for disposing of the shape when you no longer need it. You can do this using the `GXDisposeShape` function, which decrements the owner count of the shape and frees the memory occupied by that shape if the shape’s owner count becomes 0. The examples of this section dispose of the point shape by calling

```
GXDisposeShape(aPointShape);
```

Since the `GXNewPoint`, `GXNewShapeVector`, and `GXNewShape` functions all return a shape with an owner count of 1, calling the `GXDisposeShape` function in the three previous examples would decrement the owner count to 0 and therefore purge the point shape from memory. For a complete discussion of creating and disposing of shapes, see *Inside Macintosh: QuickDraw GX Objects*.

For more information about point shapes, see “Point Shapes” on page 2-16 and “The Point Structure” on page 2-104.

For information about the functions you can use to create and draw points, see the description of the `GXNewPoint` function on page 2-111 and the `GXDrawPoint` function on page 2-158.

Creating and Drawing Lines

You can draw lines and create line shapes with QuickDraw GX in much the same way as you draw points and create point shapes. Typically, you first define a line geometry, which is encapsulated in a `gxLine` structure:

```
struct gxLine {
    struct gxPoint first;
    struct gxPoint last;
};
```


Geometric Shapes

Once you've defined a line geometry, you can draw the corresponding line without creating a line shape by using the `GXDrawLine` function, as shown in Listing 2-6.

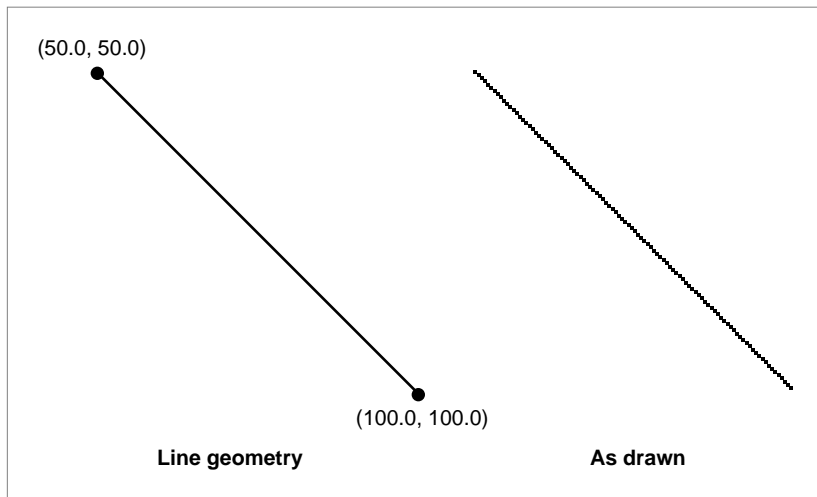
Listing 2-6 Drawing a line without creating a line shape

```
void DrawASingleLine(void)
{
    static gxLine aLineGeometry = {{ff(50), ff(50)},
                                   {ff(150), ff(150)}};

    GXDrawLine(&aLineGeometry);
}
```

This sample function defines a line geometry, using the `ff` macro (which is an alias for the `GXIntToFixed` macro) to convert integer constants to fixed-point coordinate values. It then uses the `GXDrawLine` function to draw the line. The `GXDrawLine` function uses the style, ink, and transform information from the default line shape when drawing the line. The result is shown in Figure 2-19.

Figure 2-19 A line



Geometric Shapes

As with the point shape in Figure 2-17, the line shape in Figure 2-19 is infinitely thin, but is drawn one-pixel wide—a hairline—because the default value of the pen width property of the style object is 0, which indicates that QuickDraw GX should draw the line at the thinnest perceivable resolution.

Another method of drawing a line is to encapsulate the line geometry in a line shape and then use the `GXDrawShape` function to draw the line. This method allows you to specify different style, ink, and transform information for the line. The sample function in Listing 2-7 uses this method: it creates a line shape using the `GXNewLine` function and then draws the line using the `GXDrawShape` function.

Listing 2-7 Creating a line shape with the `GXNewLine` function

```
void CreateLineShape(void)
{
    gxShape      aLineShape;

    static gxLine aLineGeometry = {ff(50), ff(50),
                                   ff(150), ff(150)};

    aLineShape = GXNewLine(&aLineGeometry);

    GXDrawShape(aLineShape);

    GXDisposeShape(aLineShape);
}
```

You can also use the `GXNewShape` or `GXNewShapeVector` functions to create a line shape. For example, to create the same line shape using the `GXNewShape` function, you could replace this line of code in the previous example:

```
aLineShape = GXNewLine(&aLineGeometry);
```

with these lines of code:

```
aLineShape = GXNewShape(gxLineType);
GXSetLine(aLineShape, &aLineGeometry);
```

In either case, the line shape would be the same, and would appear as shown in Figure 2-19.

The sample function in Listing 2-8 shows how to use the `GXSetLine` function to change the geometry of an existing line shape.

Listing 2-8 Drawing two parallel lines

```

void DrawParallelLines(void)
{
    gxShape      aLineShape;

    static gxLine aLineGeometry = {ff(50), ff(50),
                                   ff(57), ff(100)};

    static gxLine anotherLineGeometry = {ff(60), ff(50),
                                         ff(67), ff(100)};

    aLineShape = GXNewShape(gxLineType);
    GXSetLine(aLineShape, &aLineGeometry);

    GXDrawShape(aLineShape);

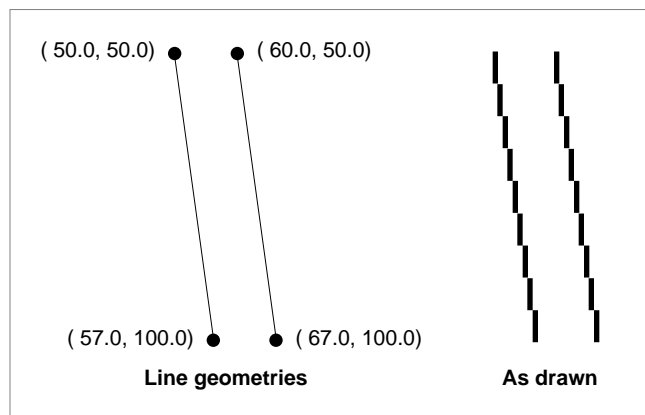
    GXSetLine(aLineShape, &anotherLineGeometry);

    GXDrawShape(aLineShape);

    GXDisposeShape(aLineShape);
}

```

This sample function creates and draws a line shape, changes its geometry, and then draws it again. The results are shown in Figure 2-20.

Figure 2-20 Parallel lines

Geometric Shapes

As with any geometric shape, you can specify fractional values for a line shape's geometric points. Although specifying a fractional part does not move the start pixel or the end pixel of line (unless rounding occurs), it can affect how the line is drawn. When QuickDraw GX draws a line with fractional endpoint coordinates, rather than integer endpoint coordinates, it may choose different pixels to represent the line, even if the endpoints remain on the same pixels in both cases. By choosing a different “stair step” pattern to represent the line, QuickDraw GX can give the illusion of very slight changes in line angles. As an example, if in the previous example you replace the second definition:

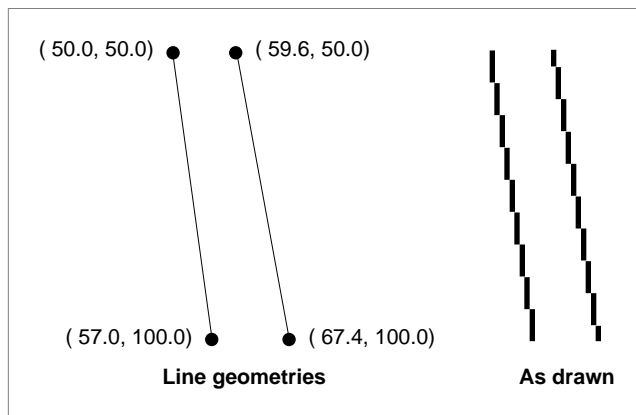
```
static gxLine anotherLineGeometry = {ff(60), ff(50),
                                     ff(67), ff(100)};
```

with a slightly modified version:

```
static gxLine anotherLineGeometry = {fl(59.6), ff(50),
                                     fl(67.4), ff(100)};
```

QuickDraw GX chooses different pixels to represent the second line, giving the appearance of a slightly different angle, as shown in Figure 2-21.

Figure 2-21 Nearly parallel lines



For more information about line shapes, see “Line Shapes” on page 2-17 and “The Line Structure” on page 2-105.

For information about the functions you can use to create and draw lines, see the description of the `GXNewLine` function on page 2-112 and the `GXDrawLine` function on page 2-158.

Creating and Drawing Curves

You can create and draw curve shapes with QuickDraw GX the same way you create and draw points and lines. Typically, you first define a curve geometry, which is encapsulated in a `gxCurve` structure:

```
struct gxCurve {
    struct gxPoint first;
    struct gxPoint control;
    struct gxPoint last;
};
```

The `first` and `last` fields determine the start point and the end point of the curve. The point specified in the `control` field lies off the curve and determines the tangents of the curve. (The off-curve control point could actually be on the curve—that is, directly between the first and last points—in which case the curve is a straight line.)

Once you've defined a curve geometry, you can create a curve shape using the `GXNewCurve` function and draw it using the `GXDrawShape` function, as shown in Listing 2-9.

Listing 2-9 Creating a curve shape

```
void CreateCurve(void)
{
    gxShape aCurveShape;

    static gxCurve aCurveGeometry = {ff(50), ff(50), /* on */
                                     ff(100), ff(150), /* off */
                                     ff(200), ff(50)}; /* on */

    aCurveShape = GXNewCurve(&aCurveGeometry);

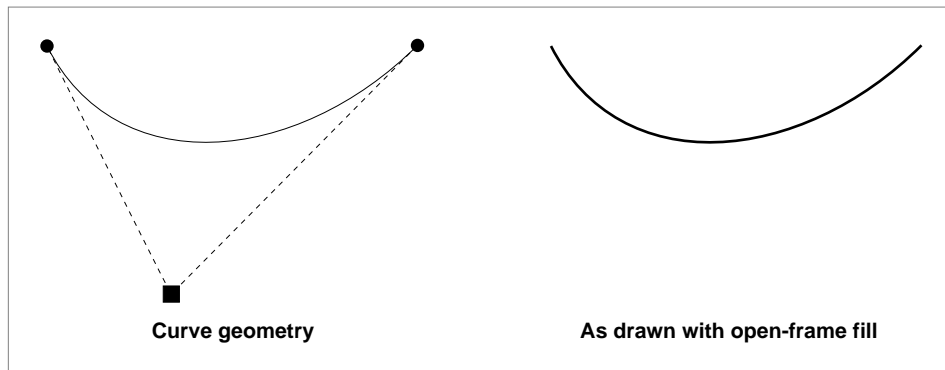
    GXDrawShape(aCurveShape);

    GXDisposeShape(aCurveShape);
}
```

Geometric Shapes

Figure 2-22 shows the curve shape geometry, which includes the first and last points, the off-curve control point, and the tangents implied by these geometric points. This figure also shows the curve as drawn. It is drawn as a hairline (one-pixel wide) with the open-frame shape fill, which reflects the default values for curve shapes.

Figure 2-22 A curve



You could draw the same curve without creating a curve shape by calling the `GXDrawCurve` function:

```
GXDrawCurve(&aCurveGeometry);
```

You could also create the curve shape using the `GXNewShape` function described in *Inside Macintosh: QuickDraw GX Objects* or the `GXNewShapeVector` function described on page 2-109.

Curves have a direction that depends on the order of the points in the geometry. For example, you could reverse the direction of the curve in Figure 2-22 by reversing the order of the points in the geometry definition from Listing 2-9:

```
static gxCurve aCurveGeometry = {ff(200), ff(50), /* on curve */
                                ff(100), ff(150), /* off curve */
                                ff(50), ff(50)}; /* on curve */
```

Changing the direction of this curve would not change its appearance. However, curve direction can affect the appearance of a curve when you apply certain stylistic variations, such as dashing, to the curve. The next chapter, “Geometric Styles,” discusses these stylistic variations. Also, when a curve is part of a path shape, the direction of the curve can affect the way the path is drawn. See “Creating and Drawing Paths” beginning on page 2-55 for examples of how the direction of a curve can affect drawing.

For more information about curve shapes, see “Curve Shapes” on page 2-18 and “The Curve Structure” on page 2-105. For information about the functions you can use to create and draw curves, see the description of the `GXNewCurve` function on page 2-113 and the `GXDrawCurve` function on page 2-159.

Creating and Drawing Rectangles

You can create rectangle shapes and draw rectangles with QuickDraw GX the same way you create and draw points, lines, and curves. Typically, you first define a rectangle geometry, which is encapsulated in a `gxRectangle` structure:

```
struct gxRectangle {
    Fixed    left;
    Fixed    top;
    Fixed    right;
    Fixed    bottom;
};
```

Note

QuickDraw GX allows you to specify rectangle coordinates out of order—that is, you can specify any corner of the rectangle using the first two fields of the rectangle structure and the opposing corner using the third and fourth fields of the rectangle structure. ♦

Once you’ve defined a rectangle geometry, you can draw the corresponding rectangle without creating a rectangle shape by using the `GXDrawRectangle` function or you can create a rectangle shape and draw it with the `GXDrawShape` function, as shown in Listing 2-10.

Listing 2-10 Creating a rectangle shape

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

    static gxRectangle aRectangleGeometry = {ff(50), ff(50),
                                              ff(150), ff(100)};

    aRectangleShape = GXNewRectangle(&aRectangleGeometry);

    GXDrawShape(aRectangleShape);

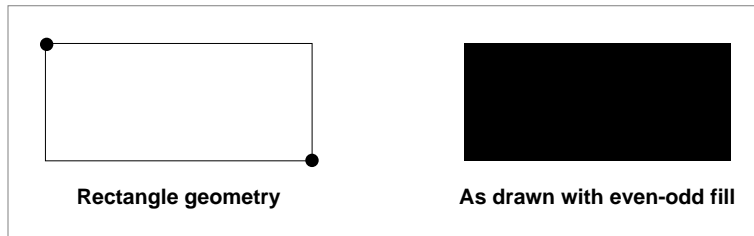
    GXDisposeShape(aRectangleShape);
}
```

This sample function uses the `ff` macro (which is an alias for the `IntegerToFixed` macro) to convert integer constants to the fixed-point coordinate values needed to define a rectangle geometry. It then creates a rectangle shape using the `GXNewRectangle` function (although it could use the `GXNewShape` function or

Geometric Shapes

the `GXNewShapeVector` function instead) and draws the rectangle using the `GXDrawShape` function. The result is shown in Figure 2-23.

Figure 2-23 A rectangle



Notice that the rectangle is solid rather than framed. The `GXNewRectangle` function returns a new rectangle shape with the same shape fill property as the default rectangle shape, which is the even-odd shape fill.

Note

Although initially QuickDraw GX sets the shape fill property of the default rectangle shape to be even-odd shape fill, you may change the default shape fill for rectangles by using the `GXGetDefaultShape` function to obtain a reference to the default rectangle and then using the `GXSetShapeFill` function to change its shape fill. You can similarly change the default shape fill for any shape type. ♦

To create a framed rectangle, you can use the `GXSetShapeFill` function to change the shape fill from even-odd to closed-frame, as shown in Listing 2-11.

Listing 2-11 Creating a framed rectangle

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

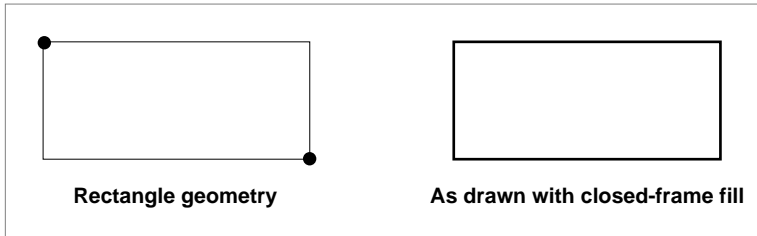
    static gxRectangle aRectangleGeometry = {ff(150), ff(100),
                                             ff(50), ff(50)};

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

    GXDrawShape(aRectangleShape);
}
```


Figure 2-24 shows the result of Listing 2-11.

Figure 2-24 A framed rectangle



In general, a rectangle can have any shape fill except open-frame shape fill. For more information about rectangle shapes, see “Rectangle Shapes” on page 2-20 and “The Rectangle Structure” on page 2-106.

For more information about the shape fill property, see “Shape Fill” beginning on page 2-12.

For information about the functions you can use to create and draw rectangles, see the description of the `GXNewRectangle` function on page 2-114 and the `GXDrawRectangle` function on page 2-160.

Creating and Drawing Polygons

A polygon contour is a series of points connected by straight lines. QuickDraw GX defines the `gxPolygon` structure to encapsulate a polygon contour:

```
struct gxPolygon {
    long          vectors;
    struct gxPoint vector[gxAnyNumber];
};
```

The `vectors` field indicates the number of points in the polygon and the vector array contains the points themselves. (The constant `gxAnyNumber` is used as a placeholder, since a polygon contour can have any number of geometric points.)

The polygon shape type allows you to group any number of polygon contours within a single QuickDraw GX shape. The `gxPolygons` structure encapsulates the multiple-polygon geometry:

```
struct gxPolygons {
    long          contours;
    struct gxPolygon contour[gxAnyNumber];
};
```

Geometric Shapes

The `contours` field indicates the total number of contours (in other words, the total number of separate polygons), and the `contour` array contains the polygon contour geometries.

Implementation Note

In version 1.0 of QuickDraw GX, a single path contour can have between 0 and 32,767 geometric points. The geometry of a path shape can between 0 and 32,767 polygon contours. The total size of a path geometry may not exceed 2,147,483,647 bytes. ♦

Creating Polygons With a Single Contour

Since a `gxPolygons` structure is of variable length and every element in it is of type `long`, you can define a polygon geometry as an array of `long` values. For example, the definition

```
long  aPolygonGeometry[] = {1, /* number of contours */
                           3, /* number of points */
                           ff(50), ff(50),
                           ff(100), ff(80),
                           ff(50), ff(110)};
```

defines a polygon geometry with one contour (that is, with one polygon). The polygon contains three points; it is a triangle.

Most QuickDraw GX functions that create or draw polygon shapes expect a pointer to a `gxPolygons` structure as one of the parameters. Therefore, you must cast an array of `long` values to the correct type before sending it to one of these functions. As an example, you can cast the `aPolygonGeometry` array to the correct type with this expression:

```
(gxPolygons *) aPolygonGeometry
```

The sample function in Listing 2-12 shows how to use this geometry to draw a triangle.

Listing 2-12 Drawing a triangular polygon

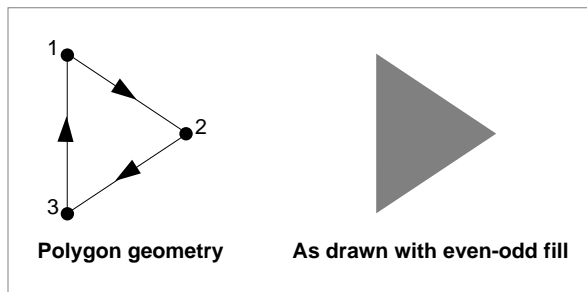
```
void DrawTriangle(void)
{
    static long aPolygonGeometry[] = {1, /* number of contours */
                                      3, /* number of points */
                                      ff(50), ff(50),
                                      ff(100), ff(80),
                                      ff(50), ff(110)};

    GXDrawPolygons((gxPolygons *) aPolygonGeometry, gxEvenOddFill);
}
```

Geometric Shapes

This sample function defines the `aPolygonGeometry` array, casts it to a `gxPolygons` pointer, and sends it to the `GXDrawPolygons` function. Unlike the `GXDrawPoint`, `GXDrawLine`, `GXDrawCurve`, and `GXDrawRectangle` functions, the `GXDrawPolygons` function takes a second parameter—the shape fill to use when drawing the polygon shape. In this example, the parameter is set to the even-odd shape value and the resulting polygon is shown in Figure 2-25.

Figure 2-25 A polygon

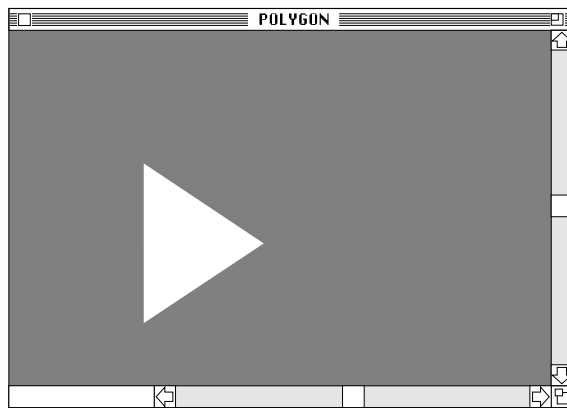


You can specify any type of shape fill for polygon shapes. For example, if you specify the inverse even-odd shape fill:

```
GXDrawPolygons((gxPolygons *) aPolygonGeometry,  
               gxInverseEvenOddFill);
```

QuickDraw GX draws the graphic shown in Figure 2-26. The black portion of the drawing would be clipped according to the information in the default polygon shape's transform object. If no clipping information is specified there, the drawing would extend to the full range of the shape's view port.

Figure 2-26 A triangular polygon with inverse shape fill



Geometric Shapes

For information on clipping and view ports, see *Inside Macintosh: QuickDraw GX Objects*.

Although this example draws the polygon without creating a polygon shape, it could instead create a polygon shape with the `GXNewPolygons` function:

```
aPolygonShape = GXNewPolygons((gxPolygons *) aPolygonGeometry);
```

and then draw it using the `GXDrawShape` function:

```
GXDrawShape(aPolygonShape);
```

You can also create polygon shapes using the `GXNewShape` function:

```
aPolygonShape = GXNewShape(gxPolygonType);
GXSetPolygons(aPolygonShape, (gxPolygons *) aPolygonGeometry);
```

or by using the `GXNewShapeVector` function:

```
aPolygonShape = GXNewShapeVector(gxPolygonType, aPolygonGeometry);
```

Notice that in this case you do not have to cast the `aPolygonGeometry` array to be a pointer to a `gxPolygons` structure. The `GXNewShapeVector` function expects an array of long values.

Although the `GXDrawPolygons` function (shown in Listing 2-12) allows you to specify a shape fill, the `GXDrawShape` function does not. If you create a polygon shape and you want it to have a different shape fill than the default polygon shape, you must indicate the desired shape fill using the `GXSetShapeFill` function—for example,

```
GXSetShapeFill(aPolygonShape, gxInverseEvenOddFill);
```

For more information about shape fills, see “Shape Fill” beginning on page 2-12.

Creating Polygons With Multiple Contours

The sample function in Listing 2-13 shows how a single polygon shape can contain more than one polygon contour. The polygon shape defined in this example includes the triangle from the previous example as well as a second, entirely separate, triangle.

Listing 2-13 Creating a polygon with two contours

```
void DrawTwoTriangles(void)
{
    gxShape      aPolygonsShape;

    static long aPolygonsGeometry[] = {2, /* number of contours */
                                        3, /* number of points */
                                        ff(50), ff(50),
                                        ff(100), ff(80),
                                        ff(50), ff(110),
                                        3, /* number of points */
                                        ff(200), ff(50),
                                        ff(150), ff(80),
                                        ff(200), ff(110)};

    aPolygonsShape = GXNewPolygons((gxPolygons *)
                                   aPolygonsGeometry);

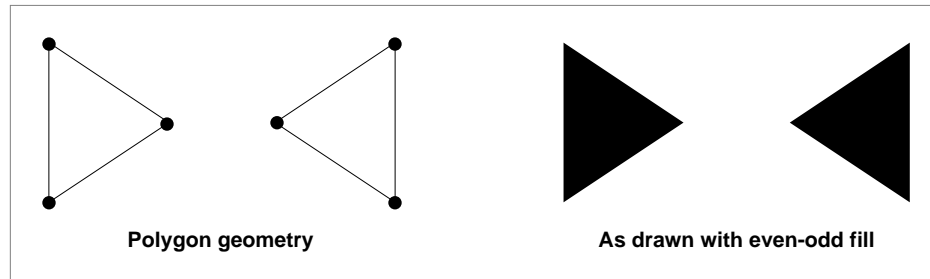
    GXDrawShape(aPolygonsShape);

    GXDisposeShape(aPolygonsShape);
}
```

Geometric Shapes

This sample function results in the drawing shown in Figure 2-27.

Figure 2-27 A filled polygon with two separate contours



For more information about polygon shapes and multiple contours, see “Polygon Shapes” beginning on page 2-22.

Creating Polygons With Crossed Contours

Since a polygon contour is defined as an array of geometric points connected by straight lines, it is possible for the lines that make up a polygon contour to cross over each other. The sample function in Listing 2-14 creates such a polygon.

Listing 2-14 Creating a polygon with a crossed contour

```
void CreateCrossedContour(void)
{
    gxShape      aPolygonsShape;

    static long aPolygonsGeometry[] = {1, /* number of contours */
                                        4, /* number of points */
                                        ff(50), ff(50),
                                        ff(150), ff(110),
                                        ff(150), ff(50),
                                        ff(50), ff(110)};

    aPolygonsShape = GXNewPolygons((gxPolygons *)
                                   aPolygonsGeometry);
    GXSetShapeFill(aPolygonsShape, gxClosedFrameFill);
}
```

Geometric Shapes

```

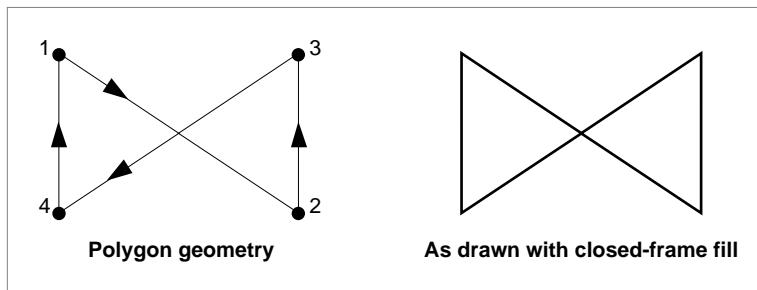
GXDrawShape(aPolygonsShape);

GXDisposeShape(aPolygonsShape);
}

```

Figure 2-28 shows the geometry of the resulting polygon contour as well as how the contour appears when drawn with the closed-frame shape fill.

Figure 2-28 A framed polygon with a crossed contour



You can change the shape fill of this polygon by removing this line of code from the sample function in Listing 2-14:

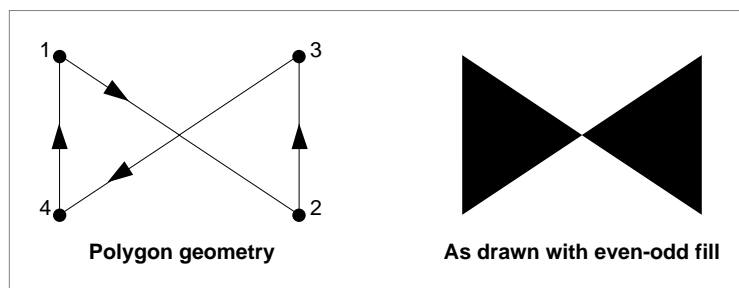
```

GXSetShapeFill(aPolygonsShape, gxClosedFrameFill);

```

If you don't specify a shape fill, the `GXNewPolygons` function uses the shape fill from the default polygon, which is the even-odd shape fill (unless you change it using the `GXGetDefaultShape` and `GXSetShapeFill` functions). The polygon resulting from an even-odd shape fill is shown in Figure 2-29.

Figure 2-29 A solid polygon with a crossed contour



Notice that QuickDraw GX fills both sections of this polygon.

Geometric Shapes

It is possible to create a polygon with a contour that overlaps in such a way that QuickDraw GX does not fill all sections of the polygon. The sample function in Listing 2-15 creates such a polygon.

Listing 2-15 Creating a polygon with an overlapping contour

```
void CreateOverlappingContour(void)
{
    gxShape      aPolygonShape;

    static long aPolygonGeometry[] = {1, /* number of contours */
                                       6, /* number of points */
                                       ff(50), ff(50),
                                       ff(100), ff(80),
                                       ff(25), ff(150),
                                       ff(25), ff(10),
                                       ff(100), ff(80),
                                       ff(50), ff(110)};

    aPolygonShape = GXNewPolygons((gxPolygons *) aPolygonGeometry);
    GXSetShapeFill(aPolygonShape, gxHollowFill);

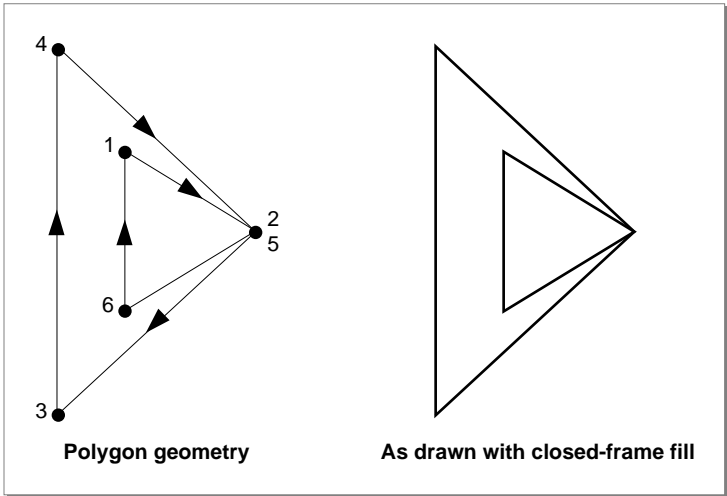
    GXDrawShape(aPolygonShape);

    GXDisposeShape(aPolygonShape);
}
```


Geometric Shapes

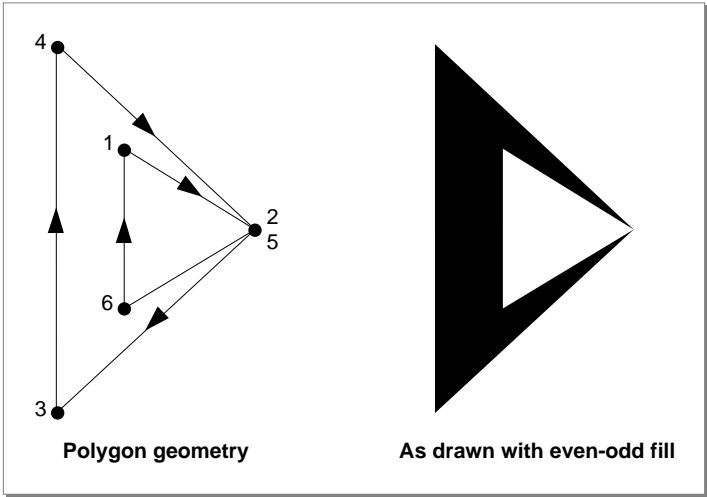
Figure 2-30 shows the geometry of the resulting polygon contour as well as how the contour appears when drawn with the closed-frame shape fill.

Figure 2-30 A polygon with an overlapping contour and closed-frame shape fill



If you specified the even-odd shape fill for this polygon, instead of the closed-frame shape fill, the resulting shape would appear as in Figure 2-31.

Figure 2-31 A polygon with an overlapping contour and even-odd shape fill



Geometric Shapes

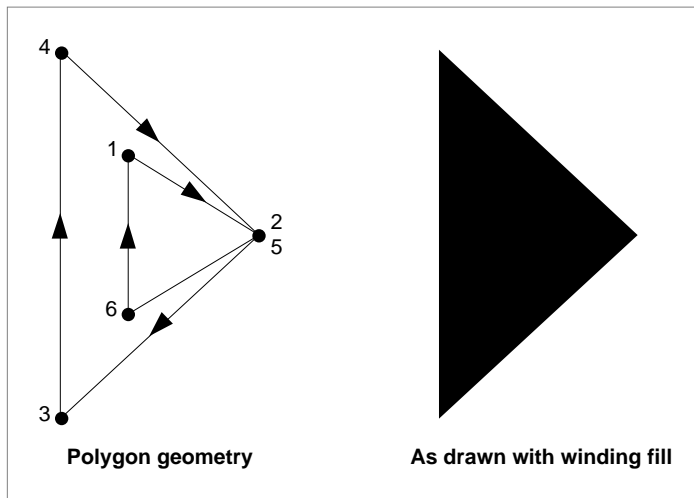
Notice that QuickDraw GX fills in the polygon but does not fill in the area contained in the inner loop. The algorithm used by QuickDraw GX to fill in shapes with the even-odd shape fill doesn't fill loops within the shape. (It would, however, fill another loop inside the first loop.)

The winding shape fill works differently. If you specify the winding shape fill for this polygon using the call

```
GXSetShapeFill(aPolygonShape, gxWindingFill);
```

QuickDraw GX draws the polygon as shown in Figure 2-32.

Figure 2-32 A polygon with an overlapping contour and winding shape fill



As you can see, the winding shape fill causes QuickDraw GX to hide the inner loop—it fills in the entire polygon, outer loop and inner.

Geometric Shapes

It is possible, however, to define a polygon in such a way that QuickDraw GX does not fill the inner loop even when you specify the winding shape fill. Unlike the even-odd shape fill, which never fills an inner loop, winding shape fill considers contour direction when filling a shape:

- If the inner loop and the outer loop have the same contour direction, winding shape fill causes QuickDraw GX to fill the inner loop as well as the outer loop, as shown in Figure 2-32.
- If the inner loop and the outer loop have opposite contour directions, winding shape fill causes QuickDraw GX to fill the outer loop, but not the inner loop. The next section gives an example using path shapes.

For more information about contour direction and shape-filling algorithms, see “Shape Fill” beginning on page 2-12.

For more information about polygon shapes, see “Polygon Shapes” on page 2-22 and “Polygon Structures” on page 2-106.

For information about the functions you can use to create and draw polygons, see the description of the `GXNewPolygons` function on page 2-116 and the `GXDrawPolygons` function on page 2-161.

Creating and Drawing Paths

Like a polygon contour, a path contour is a series of connected points. However, whereas a polygon contour is made up of straight lines, a path contour can contain both straight lines and curves. Therefore, the geometric points that make up a path contour can be on-curve points or off-curve control points. QuickDraw GX defines the `gxPath` structure to encapsulate a path contour geometry:

```
struct gxPath {
    long          vectors;
    long          controlBits[gxAnyNumber];
    struct gxPoint vector[gxAnyNumber];
};
```

Geometric Shapes

The `vectors` field indicates the number of geometric points in the path and the `vector` array contains the geometric points themselves. The `controlBits` array specifies which geometric points are on-curve points and which are off-curve control points. A value of 0 indicates an on-curve point and a value of 1 indicates an off-curve point. For example, a `controlBits` field with the value

```
0x55555555    /* 0101 0101 0101 0101 ... */
```

indicates that every other point is an off-curve control point; the first point is on curve, the second point is off, and so on. As another example, a `controlBits` field value of

```
0x00000000    /* 0000 0000 0000 0000 ... */
```

indicates all points are on curve, which effectively creates a polygon.

Notice that the `controlBits` array allows you to specify sequential off-curve control points. For example, a `controlBits` value of

```
0xFFFFFFFF    /* 1111 1111 1111 1111 ... */
```

indicates that all points are off curve. When you indicate that two control points in a row are off curve, QuickDraw GX assumes an on-curve point midway between them. (The example in Listing 2-17 on page 2-59 gives an example.)

Like the polygon shape, the path shape allows you to group any number of contours within a single QuickDraw GX shape. The `gxPaths` structure encapsulates the multiple-path geometry:

```
struct gxPaths {
    long          contours;
    struct gxPath contour[gxAnyNumber];
};
```

The `contours` field indicates the total number of contours (in other words, the total number of separate paths), and the `contour` array contains the path geometries.

Creating Paths With a Single Contour

Since a `gxPaths` structure is of variable length and every element in it is of type `long`, you can define a path geometry as an array of long values. The sample function in Listing 2-16 shows how to define a path geometry as an array of long values, and then draw a path shape using the `GXDrawPaths` function. Since the `GXDrawPaths` function expects its first parameter to be a pointer to a `gxPaths` structure, the sample function casts the array of long values to the appropriate type using the expression

```
(gxPaths *) aPathGeometry
```

before sending the information to the `GXDrawPaths` function.

Listing 2-16 Drawing a path shape

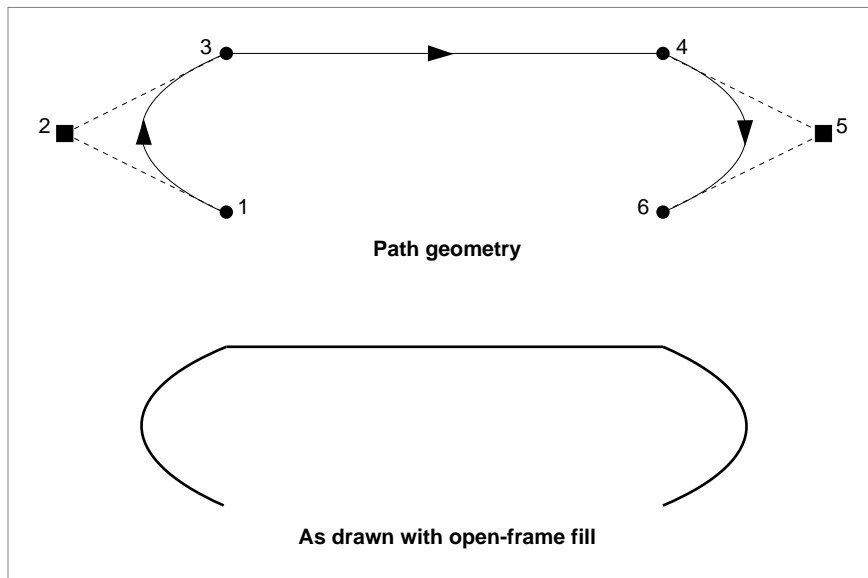
```
void DrawAPathShape(void)
{
    static long aPathGeometry[] = {1, /* number of contours */
                                    6, /* number of points */
                                    0x48000000, /* 0100 1000 */
                                    ff(50), ff(100), /* on */
                                    ff(0), ff(75), /* off */
                                    ff(50), ff(50), /* on */
                                    ff(150), ff(50), /* on */
                                    ff(200), ff(75), /* off */
                                    ff(150), ff(100)}; /* on */

    GXDrawPaths((gxPaths *) aPathGeometry, gxOpenFrameFill);
}
```

Geometric Shapes

The path defined in this example has four on-curve points and two off-curve points. When drawn with the open-frame shape fill, it contains two curves and one straight line, as shown in Figure 2-33.

Figure 2-33 A path



The sample function from Listing 2-16 draws the path without creating a path shape. It could instead create a path shape with the `GXNewPaths` function:

```
aPathsShape = GXNewPaths((gxPaths *) aPathGeometry);
```

and then draw it using the `GXDrawShape` function:

```
GXDrawShape(aPathsShape);
```

You can also create path shapes using the `GXNewShape` function:

```
aPathsShape = GXNewShape(gxPathType);
GXSetPaths(aPathsShape, (gxPaths *) aPathGeometry);
```

or by using the `GXNewShapeVector` function:

```
aPathsShape = GXNewShapeVector(gxPathType, aPathGeometry);
```

Geometric Shapes

Notice that in this case you do not have to cast the `aPathGeometry` array to be a pointer to a `gxPaths` structure. The `GXNewShapeVector` function expects an array of long values.

Although the `GXDrawPaths` function (shown in Listing 2-16) allows you to specify a shape fill, the `GXDrawShape` function does not. If you create a path shape and you want it to have a different shape fill than the default path shape, you must indicate the desired shape fill using the `GXSetShapeFill` function—for example,

```
GXSetShapeFill(aPathsShape, gxInverseEvenOddFill);
```

For more information about shape fills, see “Shape Fill” beginning on page 2-12.

Creating Paths Using Only Off-Curve Points

The sample function in Listing 2-17 shows how you can create a path using only off-curve control points. The path defined in this example contains four control points, and the `controlBits` field is set to

```
0xF0000000 /* 1111 0000 0000 0000 0000 ... */
```

which indicates that the first four points are off curve. The path contains only four points, and therefore they are all off curve.

Listing 2-17 Creating a path using only off-curve control points

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

    static long aPathGeometry[] = {1, /* number of contours */
                                   4, /* number of points */
                                   0xF0000000, /* 1111 0000 ... */
                                   ff(50), ff(50), /* off */
                                   ff(150), ff(50), /* off */
                                   ff(150), ff(150), /* off */
                                   ff(50), ff(150)}; /* off */

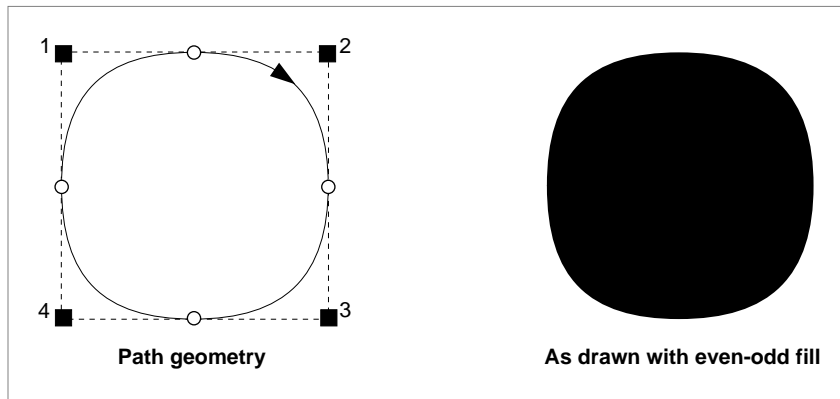
    aPathShape = GXNewPaths((gxPaths *) aPathGeometry);

    GXDrawShape(aPathShape);
}
```

Geometric Shapes

The four off-curve control points in this example form a square; the path that they define is a rounded square, as shown in Figure 2-34.

Figure 2-34 A round path shape



Notice that the path is filled with the even-odd shape fill, which is the default for path shapes. You could, however, specify any shape fill for this path except the open-frame shape fill. The open-frame shape fill requires that the first and last points of the contour be on-curve points, and this path has no on-curve points.

Creating Paths With Multiple Contours

The sample function in Listing 2-18 shows how a single path shape can contain more than one path contour. The path shape defined in this example includes the round path from the previous example as well as a second round path, entirely contained within the first.

Listing 2-18 Creating a path with concentric contours

```

void CreateHollowCircles(void)
{
    gxShape  aPathShape;

    static long aPathGeometry[] = {2, /* number of contours */
                                    4, /* number of points */
                                    0xF0000000, /* 1111 0000 ... */
                                    ff(50), ff(50), /* off */
                                    ff(150), ff(50), /* off */
                                    ff(150), ff(150), /* off */
                                    ff(50), ff(150), /* off */

                                    4, /* number of points */
                                    0xF0000000, /* 1111 0000 ... */
                                    ff(65), ff(65), /* off */
                                    ff(135), ff(65), /* off */
                                    ff(135), ff(135), /* off */
                                    ff(65), ff(135)}; /* off */

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

    GXDrawShape(aPathShape);

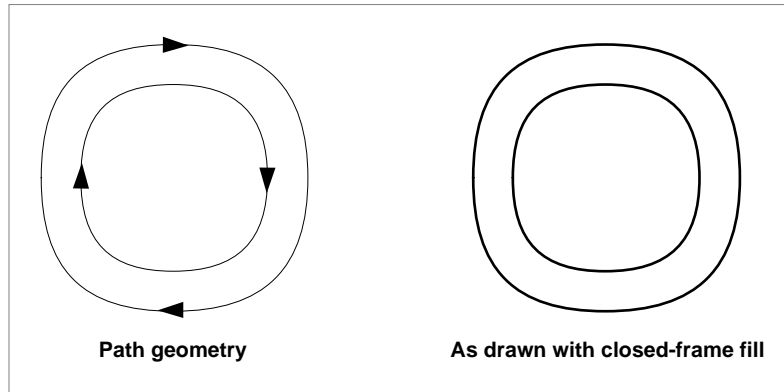
    GXDisposeShape(aPathShape);
}

```

Geometric Shapes

The result of this function is shown in Figure 2-35.

Figure 2-35 A path shape with two concentric clockwise contours and closed-frame shape fill



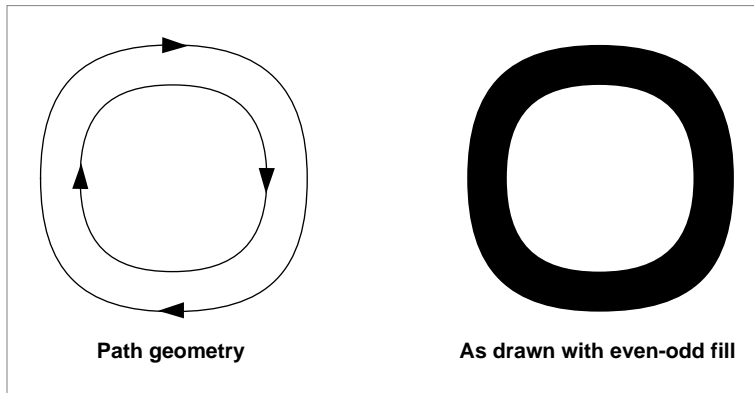
You can change the shape fill of this polygon by removing this line of code from the sample function in Listing 2-18:

```
GXSetShapeFill(aPolygonsShape, gxClosedFrameFill);
```

Geometric Shapes

If you don't specify a shape fill, the `GXNewPaths` function uses the shape fill from the default path shape, which is the even-odd shape fill (unless you change it using the `GXGetDefaultShape` and `GXSetShapeFill` functions). The path shape resulting from an even-odd shape fill is shown in Figure 2-36.

Figure 2-36 A path shape with two concentric clockwise contours and even-odd shape fill

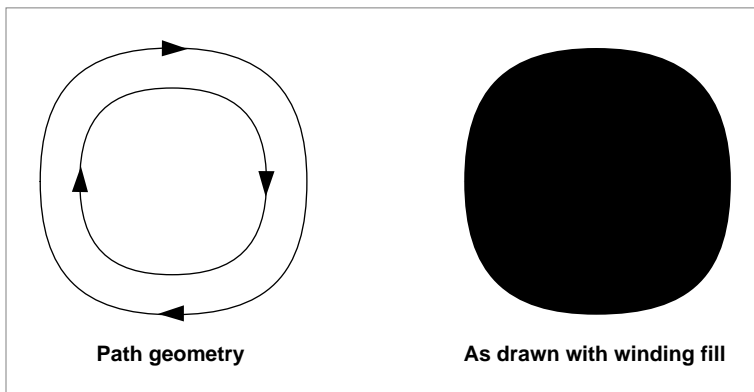


Notice that the even-odd shape fill causes QuickDraw GX to fill in the outer contour, but not the inner contour. However, if you specify the winding shape fill for this path using the call

```
GXSetShapeFill(aPathShape, gxWindingFill);
```

the resulting shape would appear as shown in Figure 2-37.

Figure 2-37 A path shape with two concentric clockwise contours and winding shape fill



Geometric Shapes

Unlike the even-odd shape fill, the winding shape fill causes QuickDraw GX to fill inner contours—as long as the inner contour has the same contour direction as the outer contour. If the inner contour and the outer contour have opposite contour directions, neither the even-odd shape fill nor the winding shape fill will fill the inner contour.

For example, if you change the direction of the inner contour from the previous example by reversing the order of the second path's geometric points, as in the declaration

```
static long aPathGeometry[] = {2, /* number of contours */
                               4, /* number of points */
                               0xF0000000, /* 1111 0000 */
                               ff(50), ff(50), /* off */
                               ff(150), ff(50), /* off */
                               ff(150), ff(150), /* off */
                               ff(50), ff(150), /* off */

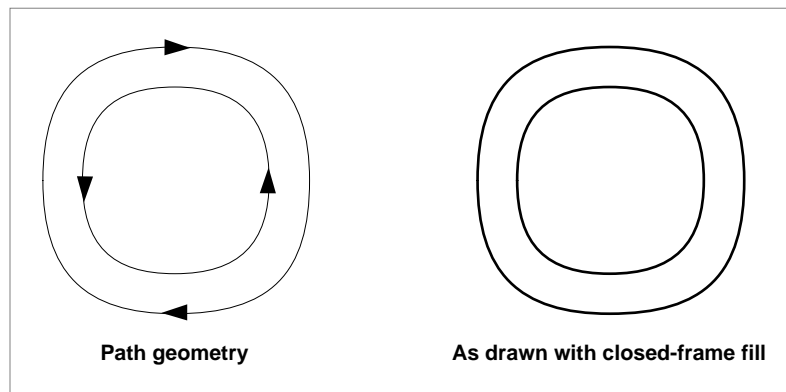
                               4, /* number of points */
                               0xF0000000, /* 1111 0000 */
                               ff(65), ff(135), /* off */
                               ff(135), ff(135), /* off */
                               ff(135), ff(65), /* off */
                               ff(65), ff(65)}; /* off */
```

and set the shape fill to the closed-frame shape fill using the call

```
GXSetShapeFill(aPathShape, gxClosedFrameFill);
```

the resulting shape has contours with opposite contour directions, as depicted in Figure 2-38.

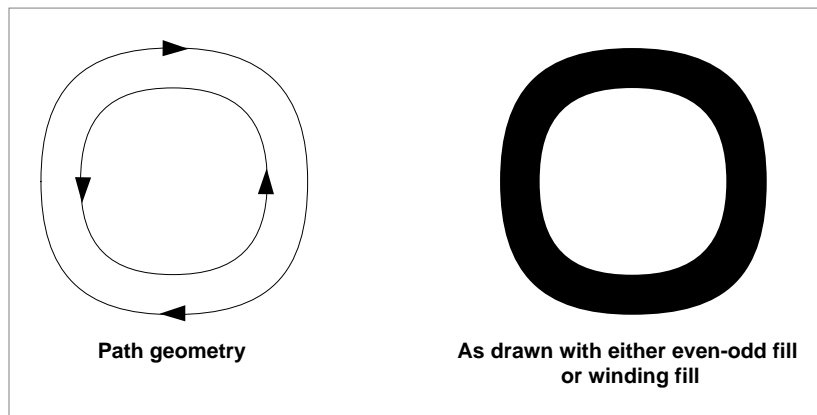
Figure 2-38 A path shape with an internal counterclockwise contour and closed-frame shape fill



Geometric Shapes

Since the outer contour and the inner contour have opposite contour directions, neither the even-odd shape fill nor the winding shape fill cause QuickDraw GX to fill the inner contour, as shown in Figure 2-39.

Figure 2-39 A path shape with even-odd or winding shape fill



For more information about contour direction and shape-filling algorithms, see “Shape Fill” on page 2-12.

For more information about path shapes, see “Path Shapes” on page 2-25 and “Path Structures” on page 2-107.

For information about the functions you can use to create and draw paths, see the description of the `GXNewPaths` function on page 2-117 and the `GXDrawPaths` function on page 2-162.

Converting Between Geometric Shape Types

QuickDraw GX provides the `GXGetShapeType` and `GXSetShapeType` functions to allow you to manipulate a shape’s type. The `GXGetShapeType` function simply returns the value of the shape type property for a specified shape. For geometric shapes, the possible values returned from this function are

- `gxEmptyType`
- `gxFullType`
- `gxPointType`
- `gxLineType`
- `gxCurveType`
- `gxRectangleType`

Geometric Shapes

- `gxPolygonType`
- `gxPathType`

The `GXSetShapeType` function allows you to change the shape type of an existing shape. In doing so, this function often has to reinterpret the geometry of the shape. This reinterpretation is called **type conversion**. Sometimes the conversion makes sense and doesn't lose any data. For example, you might want to convert a line shape to a polygon shape so that you can add more contours to the shape. Some conversions, however, aren't as useful and data can be lost. For example, converting a complex path shape to a point shape can result in the loss of a significant amount of data.

In general, when converting between geometric shape types, QuickDraw GX exhibits different behavior in these four situations:

- when converting other geometric shapes to an empty shape or a full shape
- when converting other geometric shapes to a point, line, or rectangle
- when converting other geometric shapes to a curve
- when converting other geometric shapes to a polygon or path

When converting to an empty shape or a full shape, all the information in the original shape's geometry is lost—the result is simply an empty shape or a full shape, respectively.

The following three subsections discuss the other cases in more detail.

Converting Shapes to Points, Lines, and Rectangles

When converting a shape to a point, line, or rectangle, QuickDraw GX uses the bounding rectangle of the original shape. (Bounding rectangles are defined in the chapter “Geometric Operations” in this book. For an example, see Figure 2-41 on page 2-68.) QuickDraw GX uses the bounding rectangle differently, depending on which shape type you are converting to:

- When you convert to a rectangle shape, the resulting rectangle is the bounding rectangle of the original shape.
- When you convert to a line shape, the result is a line that runs from the upper-left corner of the original shape's bounding rectangle to the lower-right corner.
- When you convert to a point, the resulting point is the point at the upper-left corner of the bounding rectangle of the original shape.

Geometric Shapes

Listing 2-19 creates a path shape, which is converted subsequently to a rectangle shape, then to a line shape, and finally to a point shape.

Listing 2-19 Creating a figure-eight path shape

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

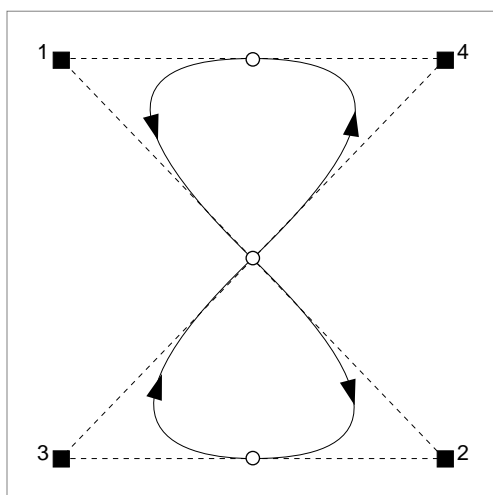
    static long figureEightGeometry[] = {1, /* number of contours */
                                         4, /* number of points */
                                         0xF0000000, /* 1111 ... */
                                         ff(50), ff(50), /* off */
                                 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

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

    GXDrawShape(aPathShape);
}
```

The resulting path geometry is shown in Figure 2-40.

Figure 2-40 A figure-eight path shape



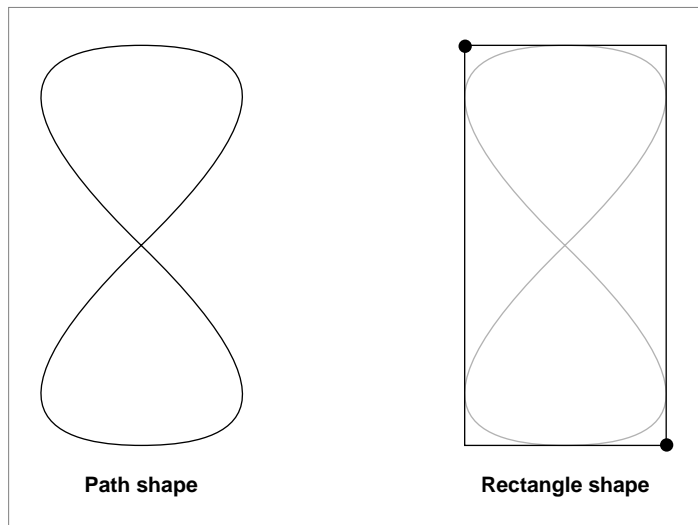
Geometric Shapes

If you convert this shape to a rectangle shape, using the call

```
GXSetShapeType(aPathShape, gxRectangleType);
```

the resulting shape is the bounding rectangle for the original path shape, as shown in Figure 2-41.

Figure 2-41 A path shape before and after conversion to a rectangle shape



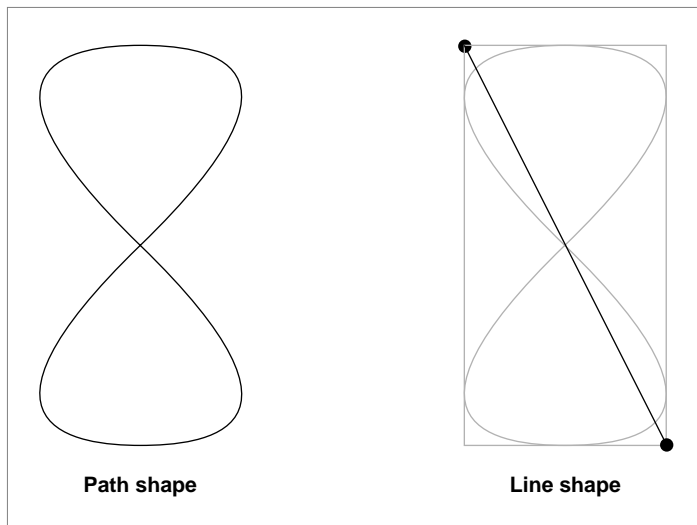
Geometric Shapes

If you convert the original path shape to a line shape, using the call

```
GXSetShapeType(aPathShape, gxLineType);
```

the resulting shape is a diagonal line from the upper-left corner of the path's bounding rectangle to its lower-right corner, as shown in Figure 2-42.

Figure 2-42 A path shape before and after conversion to a line shape



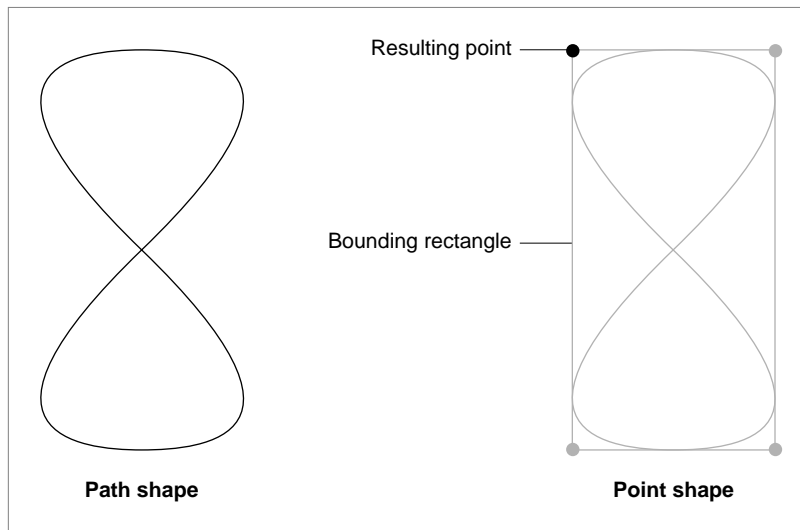
Geometric Shapes

Finally, if you convert the original path shape to a point shape, using the call

```
GXSetShapeType(aPathShape, gxPointType);
```

the resulting shape is the point at the upper-left corner of the path's bounding rectangle, as shown in Figure 2-43.

Figure 2-43 A path shape before and after conversion to a point shape



The next two sections give examples of converting to the curve shape type and of converting to the polygon and path shape types.

For more information about the `GXSetShapeType` function, see the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

Converting Shapes to Curve Shapes

When converting other geometric shapes to a curve shape, QuickDraw GX takes one of these approaches:

- When converting a point to a curve, QuickDraw GX sets each of the three geometric points of the curve to be the same as the original point.
- When converting a line to a curve, QuickDraw GX sets the first point and last point of the curve to be the same as the first point and last point of the line and sets the off-curve control point to be same as the last point of the line, which results in the curve being a straight line.
- When converting a rectangle to a curve, QuickDraw GX sets the first point of the curve to be the upper-left corner of the rectangle and the last point of the curve to be the lower-right corner of the rectangle. The off-curve control point is set to be the same as the last point, which results in the curve being a straight line.
- When converting a polygon or a path to a curve, QuickDraw GX sets the three geometric points of the curve to be the first three geometric points of the original shape.

The sample function in Listing 2-20 creates a line shape and converts it to a curve.

Listing 2-20 Converting a line to a curve

```
void ConvertLineToCurve(void)
{
    gxShape      aLineShape;

    static gxLine diagonalGeometry = {ff(50), ff(50),
                                      ff(150), ff(150)};

    aLineShape = GXNewLine(&diagonalGeometry);
    GXSetShapeType(aLineShape, gxCurveType);

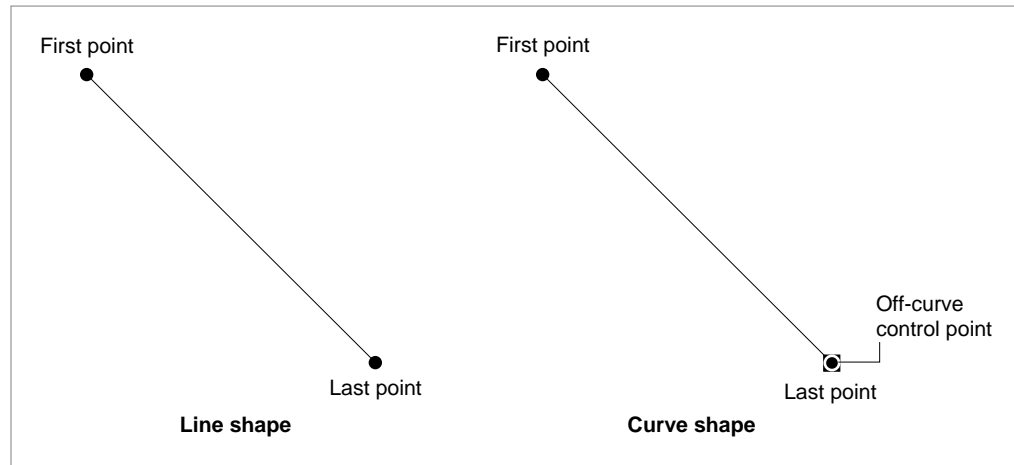
    GXDrawShape(aLineShape);

    GXDisposeShape(aLineShape);
}
```

Geometric Shapes

The original line shape and the resulting curve shape are shown in Figure 2-44.

Figure 2-44 A line shape before and after conversion to a curve shape



Notice that the converted curve looks just like the original line. The only difference between the two shapes is that the curve shape has an additional off-curve control point, which is set to be identical to the last point.

The sample function in Listing 2-21 creates a rectangle shape and converts it to a curve shape.

Listing 2-21 Converting a rectangle to a curve

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

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

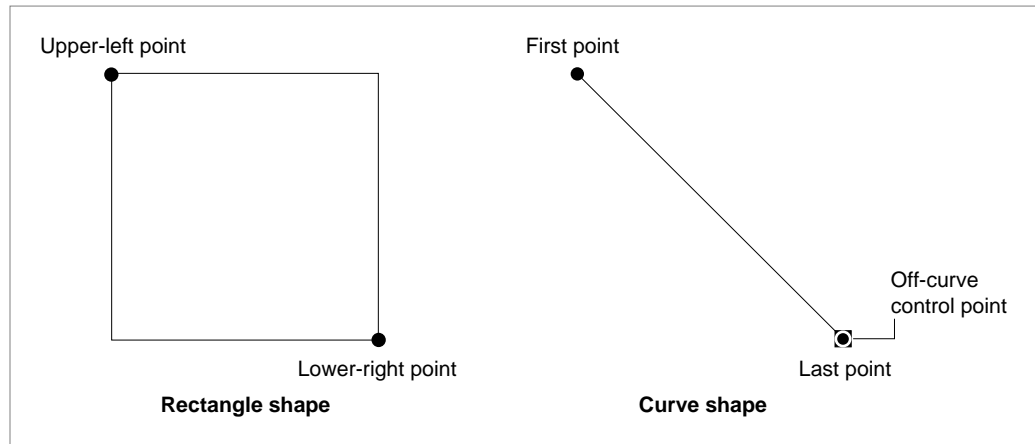
    aRectangleShape = GXNewRectangle(&rectangleGeometry);
    GXSetShapeType(aRectangleShape, gxCurveType);

    GXDrawShape(aRectangleShape);

    GXDisposeShape(aLineShape);
}
```

The original rectangle and the resulting curve are both shown in Figure 2-45.

Figure 2-45 A rectangle shape before and after conversion to a curve shape



As in the previous example, the off-curve control point of the curve shape is set to be the same as the last point, which results in the curve shape being a straight line.

The next example, shown in Listing 2-22, shows how QuickDraw GX converts a polygon shape to a curve shape.

Listing 2-22 Converting a polygon shape to a curve shape

```
void ConvertPolygonToCurve(void)
{
    gxShape aPolygonShape;

    static long aPolygonGeometry[] = {1, /* number of contours */
                                      4, /* number of points */
                                      ff(50), ff(50),
                                      ff(150), ff(50),
                                      ff(150), ff(150),
                                      ff(50), ff(150)};

    aPolygonShape = GXNewPolygons((gxPolygons *) aPolygonGeometry);
    GXSetShapeType(aPolygonShape, gxCurveType);
}
```

Geometric Shapes

```

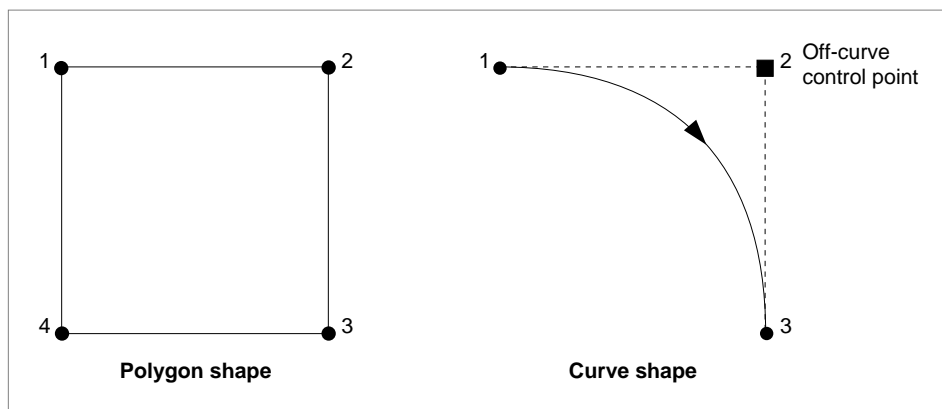
GXDrawShape(aPolygonShape);

GXDisposeShape(aPolygonShape);
}

```

In this example, QuickDraw GX sets the three geometric points of the resulting curve to be the first three geometric points of the original polygon. (Converting from path shapes to curve shapes works in the same way.) The original polygon and the resulting curve are shown in Figure 2-46.

Figure 2-46 A polygon shape before and after conversion to a curve shape



Notice that even though the polygon in this example looks the same as the rectangle in Figure 2-45, the converted curve shape looks quite different.

The next section gives examples of converting shapes to polygon and path shapes.

For more information about the `GXSetShapeType` function in general, see the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

Converting Shapes to Polygons and Paths

When converting other geometric shapes to polygon or path shapes, the original shapes don’t lose any geometric information. For example, when you convert a line shape to a path shape, the resulting path shape contains one contour with two geometric points, both on curve—an exact duplicate of the original line.

You can even convert a curve shape to a polygon shape without losing geometric information, although the result does draw differently. The resulting polygon has one contour and three geometric points—the same three geometric points as the original curve. If you convert the polygon back to a curve, you end up with the original curve.

Geometric Shapes

When you convert a rectangle shape to a polygon shape, as shown in Listing 2-23, the original shape and the resulting shape look exactly the same.

Listing 2-23 Converting a rectangle shape to a polygon shape

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

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

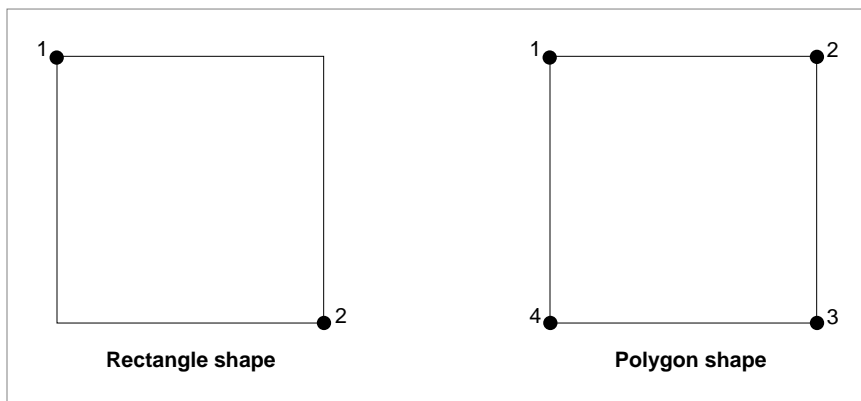
    aRectangleShape = GXNewRectangle(&rectangleGeometry);
    GXSetShapeType(aRectangleShape, gxPolygonType);

    GXDrawShape(aRectangleShape);

    GXDisposeShape(aRectangleShape);
}
```

The original rectangle and the resulting polygon are shown in Figure 2-47.

Figure 2-47 A rectangle shape before and after conversion to a polygon shape



Geometric Shapes

Converting from a path shape to a polygon shape, however, does lose geometric information. The resulting polygon contains the same geometric points as the original path; however, the points are all considered on-curve points. The original information about which points were on curve and which points were off curve is lost during this conversion.

As an example, Listing 2-24 creates a path shape and converts it to a polygon shape.

Listing 2-24 Converting a path shape to a polygon shape

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

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

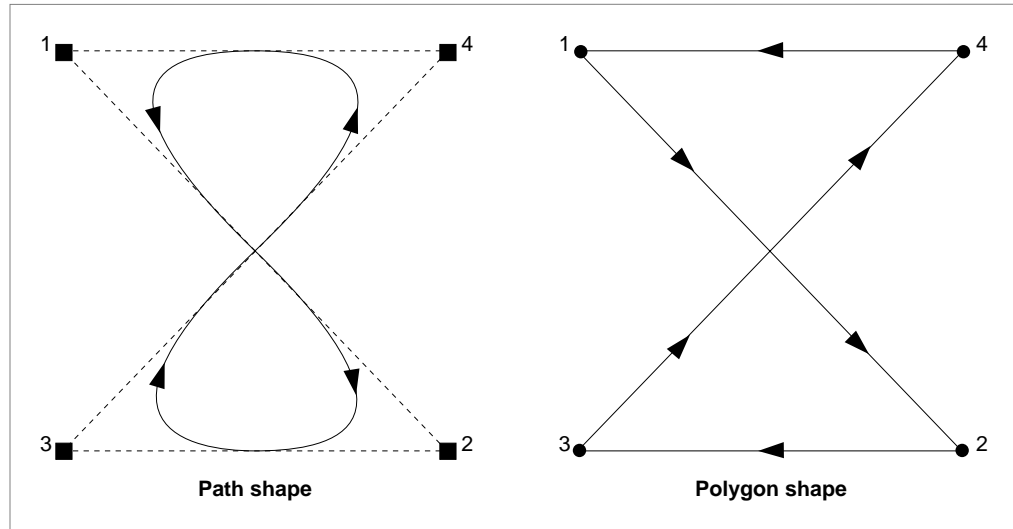
    aPathShape = GXNewPaths((gxPaths *) figureEightGeometry);
    GXSetShapeFill(aPathShape, gxHollowFill);
    GXSetShapeType(aPathShape, gxPolygonType);

    GXDrawShape(aPathShape);

    GXDisposeShape(aPathShape);
}
```


Figure 2-48 shows the original path shape and the resulting polygon shape.

Figure 2-48 A path shape before and after conversion to a polygon shape



Note

You can request that QuickDraw GX calculate a better polygon approximation to a path by setting the curve error property of the path shape's style object before calling the `GXSetShapeType` function. See the next chapter, "Geometric Styles," for examples. ♦

Geometric Shapes

Converting in the other direction, however—from a polygon shape to a path shape—retains all of the geometry information and the resulting path shape looks exactly the same as the original polygon shape. The sample function in Listing 2-25 gives an example.

Listing 2-25 Converting a polygon shape to a path shape

```
void ConvertPolygonToPath(void)
{
    gxShape      aPolygonShape;

    static long aPolygonGeometry[] = {2, /* number of contours */
                                       3, /* number of points */
                                       ff(50), ff(50),
                                       ff(100), ff(80),
                                       ff(50), ff(110),
                                       3, /* number of points */
                                       ff(200), ff(50),
                                       ff(150), ff(80),
                                       ff(200), ff(110)};

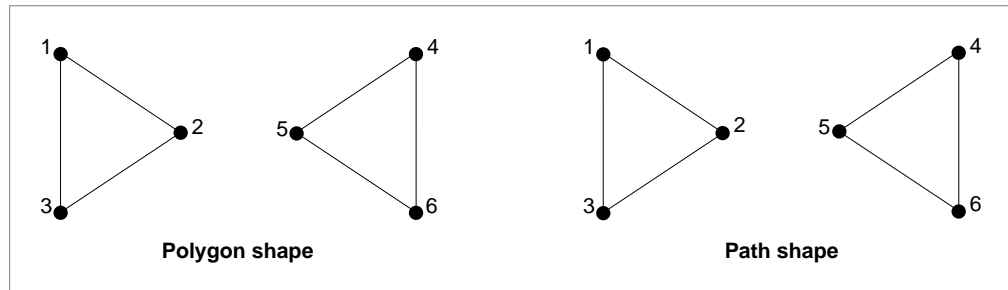
    aPolygonShape = GXNewPolygons((gxPolygons *)
                                   aPolygonGeometry);
    GXSetShapeType(aPolygonShape, gxPathType);

    GXDrawShape(aPolygonShape);

    GXDisposeShape(aPolygonShape);
}
```

The original polygon shape and the converted path shape are shown in Figure 2-49.

Figure 2-49 Polygon shape with two contours before and after conversion to a path shape



For more information about the `GXSetShapeType` function, see the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

Replacing Geometric Points

The `GXSetPoint`, `GXSetLine`, `GXSetCurve`, `GXSetRectangle`, `GXSetPolygons`, and `GXSetPaths` functions allow you to replace the geometry of an existing shape. The limitation of these functions is that you must replace the entire geometry at once.

QuickDraw GX provides other, more sophisticated, mechanisms for editing shape geometries. The `GXSetShapePoints` function, which is illustrated in this section, allows you to replace individual geometric points within a shape’s geometry. The `GXSetPolygonParts`, `GXSetPathParts`, and `GXSetShapeParts` functions, which are discussed in the next three sections, provide even more ways to edit the geometries of shapes.

Geometric Shapes

The sample function in Listing 2-26 creates a path shape and uses the `GXSetShapePoints` function to replace two of the path's geometric points.

Listing 2-26 Replacing geometric points

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

    static long twoCurveGeometry[] = {1, /* number of contours */
                                       6, /* number of points */
                                       0x48000000, /* 0100 1000 */
                                       ff(100), ff(150), /* on */
                                       ff(50),  ff(100), /* off */
                                       ff(100), ff(50),  /* on */
                                       ff(200), ff(50),  /* on */
                                       ff(250), ff(100), /* off */
                                       ff(200), ff(150)}; /* on */

    static gxPoint newTopGeometry[] = {ff(140), ff(50),
                                       ff(160), ff(50)};

    aPathShape = GXNewPaths((gxPaths *) twoCurveGeometry);
    GXSetShapeFill(aPathShape, gxOpenFrameFill);

    GXSetShapePoints(aPathShape, 3, 2, newTopGeometry);

    GXDrawShape(aPathShape);

    GXDisposeShape(aPathShape);
}
```

The `GXSetShapePoints` function takes four parameters:

- a reference to the shape to edit
- the index of the first geometric point to be replaced
- the number of geometric points to replace
- an array containing the new geometric points

Geometric Shapes

Therefore, the line of code from the sample function in Listing 2-26

```
GXSetShapePoints(aPathsShape, 3, 2, newTopGeometry);
```

replaces the third and fourth geometric point from the original path shape with the two geometric points in the `newTopGeometry` array.

Figure 2-50 shows the path shape before the geometric points are replaced.

Figure 2-50 A path shape with a flat top

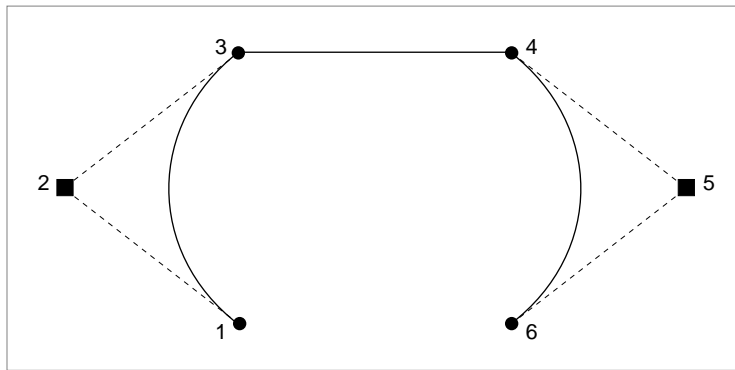
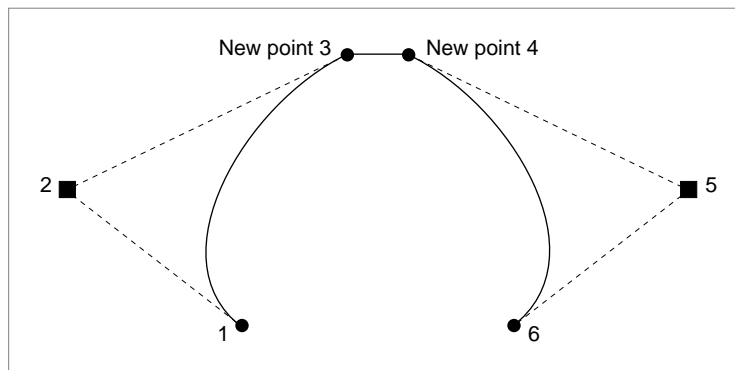


Figure 2-51 shows the path shape after the geometric points are replaced.

Figure 2-51 A path shape with geometric points replaced



For more information about the `GXSetShapePoints` function, see page 2-142.

The next three sections give examples of functions that allow you even more control in editing the geometric points of a shape's geometry.

Editing Polygon Parts

QuickDraw GX provides six functions that allow sophisticated editing of geometric shapes:

- The `GXGetPolygonParts` and `GXSetPolygonParts` functions allow you to extract information from a polygon geometry, replace information in the geometry, remove information in the geometry, and insert new information in the geometry.
- The `GXGetPathParts` and `GXSetPathParts` functions allow you to extract, replace, remove, and insert information in a path shape's geometry.
- The `GXGetShapeParts` and `GXSetShapeParts` functions allow you to extract, replace, remove, and insert information in any shape's geometry.

This section gives examples of the `GXGetPolygonParts` and `GXSetPolygonParts` functions. The next two sections show how to edit path shape geometries and shape geometries in general.

Listing 2-27 creates a polygon shape with two contours. Later examples in this section use this polygon shape to demonstrate editing polygon parts.

Listing 2-27 Creating a polygon shape with two contours

```
void CreateTwoAngles(void)
{
    gxShape      aPolygonShape;

    static long twoAngleGeometry[] = {2, /* number of contours */
                                       3, /* number of points */
                                       ff(100), ff(150),
                                       ff(50), ff(100),
                                       ff(100), ff(50),
                                       3, /* number of points */
                                       ff(200), ff(50),
                                       ff(250), ff(100),
                                       ff(200), ff(150)};

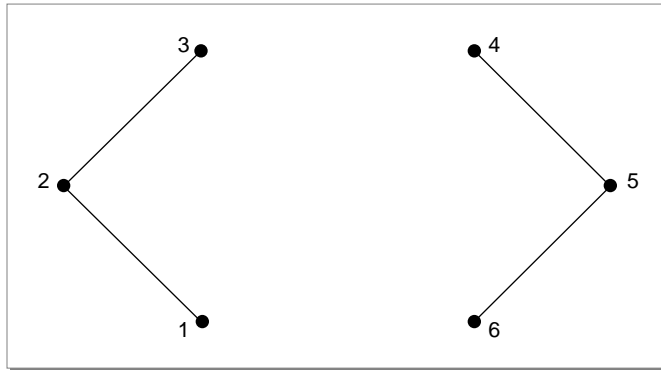
    aPolygonShape = GXNewPolygons((gxPolygons *)
                                  twoAngleGeometry);
    GXSetShapeFill(aPolygonShape, gxOpenFrameFill);

    GXDrawShape(aPolygonShape);

    GXDisposeShape(aPolygonShape);
}
```

The result of this sample function is shown in Figure 2-52.

Figure 2-52 A polygon shape with two contours



The `GXGetPolygonParts` function allows you to extract geometric points from the geometry of an existing polygon shape and put them into a new polygon geometry. This function takes four parameters:

- a reference to the existing polygon shape
- the index of the first desired geometric point
- the number of geometric points to include
- a pointer to a polygon geometry in which to store the extracted geometric points

The `GXGetPolygonParts` function returns as its function result the number of bytes necessary to contain the extracted polygon. Therefore, you typically call `GXGetPolygonParts` twice—once to determine the size of extracted polygon and once to extract the polygon. For example, if you declare the variable

```
long    byteCount;
```

you could determine the number of bytes necessary to extract the top half of the polygon geometry in Figure 2-52 using this line of code:

```
byteCount = GXGetPolygonParts(aPolygonsShape, 2, 4, nil);
```

In this example, setting the final parameter to `nil` indicates that you want to determine the number of bytes necessary to hold the extracted polygon geometry, but you do not want to actually extract the polygon geometry. The values of 2 and 4 for the second and third parameters indicate that the `GXGetPolygonParts` function should determine the number of bytes necessary to hold an extracted polygon geometry that contains geometric points 2, 3, 4, and 5 from the polygon geometry in Figure 2-52.

Geometric Shapes

You can then use this byte count to allocate enough memory to hold the extracted polygon geometry:

```
gxPolygons *topHalfGeometry;
topHalfGeometry = (gxPolygons *) NewPtr(byteCount);
```

Finally, you can extract the polygon geometry by calling `GXGetPolygonParts` again:

```
GXGetPolygonParts(aPolygonShape, 2, 4, topHalfGeometry);
```

The sample function in Listing 2-28 creates the polygon shape from the previous example, extracts the second through the fifth geometric points and puts them into a separate geometry, and then replaces the geometry of the original polygon shape with the extracted geometry.

Listing 2-28 Extracting part of a polygon shape

```
void ExtractTopPartOfPolygon(void)
{
    gxShape      aPolygonShape;

    static long twoAngleGeometry[] = {2, /* number of contours */
                                       3, /* number of points */
                                       ff(100), ff(150),
                                       ff(50), ff(100),
                                       ff(100), ff(50),
                                       3, /* number of points */
                                       ff(200), ff(50),
                                       ff(250), ff(100),
                                       ff(200), ff(150)};

    long        byteCount;

    gxPolygons *topHalfGeometry;

    aPolygonShape = GXNewPolygons((gxPolygons *)
                                   twoAngleGeometry);
    GXSetShapeFill(aPolygonShape, gxOpenFrameFill);
```


Geometric Shapes

```

byteCount = GXGetPolygonParts(aPolygonShape, 2, 4, nil);
topHalfGeometry = (gxPolygons *) NewPtr(byteCount);
GXGetPolygonParts(aPolygonShape, 2, 4, topHalfGeometry);

GXSetPolygons(aPolygonShape, topHalfGeometry);

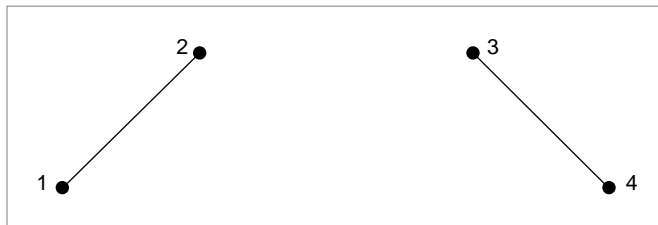
GXDrawShape(aPolygonShape);

GXDisposeShape(aPolygonShape);
}

```

The resulting polygon shape is shown in Figure 2-53.

Figure 2-53 A polygon shape extracted from a larger polygon shape



Compare this polygon shape with the polygon shape shown in Figure 2-52.

Like the `GXSetShapePoints` function discussed in the previous section, the `GXSetPolygonParts` function allows you to replace geometric points within a polygon shape's geometry. However, the `GXSetPolygonParts` function allows you even more editing control. With it, you can also remove geometric points, insert geometric points, and break a polygon shape into multiple contours.

Geometric Shapes

As an example of replacing geometric points in a polygon geometry, the sample function in Listing 2-29 creates two polygon geometries: the two-angle polygon geometry from Figure 2-52 and another polygon geometry consisting of a single point. The sample function creates the two-angle polygon shape as in Listing 2-27 and then replaces its third and fourth geometric points with the single geometric point of the other polygon geometry.

Listing 2-29 Replacing geometric points of a polygon shape

```
void ReplaceControlPoints(void)
{
    gxShape      aPolygonShape;

    static long twoAngleGeometry[] = {2, /* number of contours */
                                      3, /* number of points */
                                      ff(100), ff(150),
                                      ff(50), ff(100),
                                      ff(100), ff(50),
                                      3, /* number of points */
                                      ff(200), ff(50),
                                      ff(250), ff(100),
                                      ff(200), ff(150)};

    static long newTopGeometry[] = {1, /* number of contours */
                                    1, /* number of points */
                                    ff(150), ff(50)};

    aPolygonShape = GXNewPolygons((gxPolygons *)
                                  twoAngleGeometry);
    GXSetShapeFill(aPolygonsShape, gxOpenFrameFill);

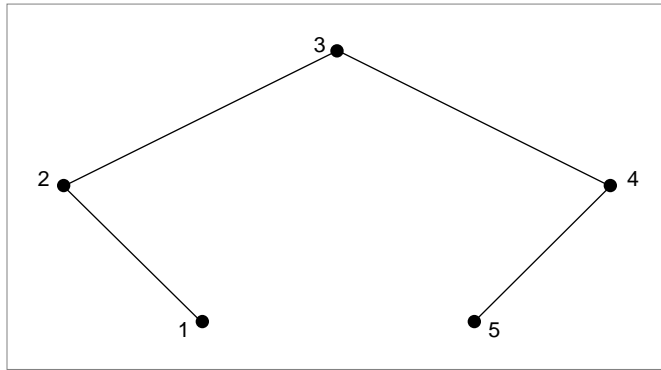
    GXSetPolygonParts(aPolygonShape, 3, 2,
                     (gxPolygons *) newTopGeometry,
                     gxBreakNeitherEdit);

    GXDrawShape(aPolygonShape);

    GXDisposeShape(aPolygonShape);
}
```

Figure 2-54 shows the result of the sample function in Listing 2-29.

Figure 2-54 A polygon with two geometric points replaced by a single geometric point

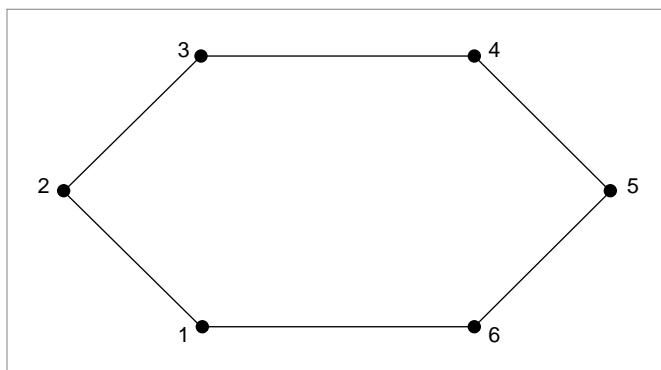


Notice that, whereas the `GXSetShapePoints` function limited you to replacing geometric points on a point-by-point basis, the `GXSetPolygonParts` function allows you to replace any number of geometric points in the original geometry with any number of new geometric points contained in an arbitrary polygon geometry.

Since the `GXSetPolygonParts` function allows you to insert an arbitrary polygon geometry into the geometry of an existing polygon shape, you can use this function to break a single polygon contour into multiple contours. In fact, the final parameter to `GXSetPolygonParts` allows you to specify how the resulting polygons shape should be broken up. In the previous example, the `gxBreakNeitherEdit` constant indicated that the resulting polygon should not be broken into separate contours.

The next example, shown in Listing 2-30, first creates a polygon shape similar to the two-angle polygons shape, except in this example the two contours are connected, as shown in Figure 2-55.

Figure 2-55 A polygon shape with one contour



Geometric Shapes

The sample function then uses the `GXSetPolygonParts` function to insert a new geometric point in the center of the polygon.

Listing 2-30 Inserting a geometric point in a polygon shape

```
void CreateHollowPolygon(void)
{
    gxShape      aPolygonShape;

    static long twoAngleGeometry[] = {1, /* number of contours */
                                      6, /* number of points */
                                      ff(100), ff(150),
                                      ff(50), ff(100),
                                      ff(100), ff(50),
                                      ff(200), ff(50),
                                      ff(250), ff(100),
                                      ff(200), ff(150)};

    static long newCenterGeometry[] = {1, /* number of contours */
                                       1 /* number of points */ ,
                                       ff(150), ff(100)};

    aPolygonShape = GXNewPolygons((gxPolygons *)
                                  twoAngleGeometry);
    GXSetShapeFill(aPolygonShape, gxClosedFrameFill);

    GXSetPolygonParts(aPolygonShape, 4, 0,
                     (gxPolygons *) newCenterGeometry,
                     gxBreakNeitherEdit);

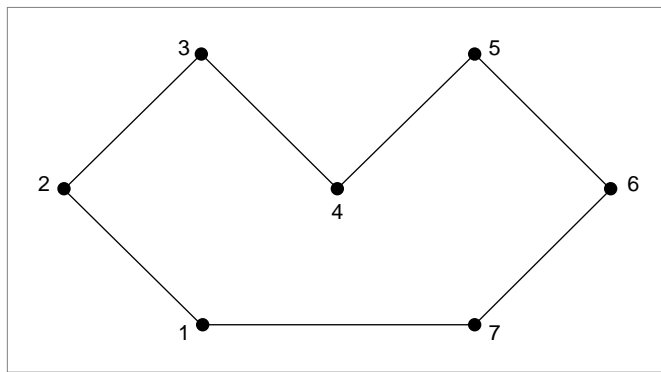
    GXDrawShape(aPolygonShape);

    GXDisposeShape(aPolygonShape);
}
```

Geometric Shapes

Since this sample function specifies the `gxBreakNeitherEdit` constant as the final parameter to the `GXSetPolygonParts` function, the resulting polygon has a single contour, as shown in Figure 2-56.

Figure 2-56 A polygon shape edited with the `gxBreakNeitherEdit` flag set

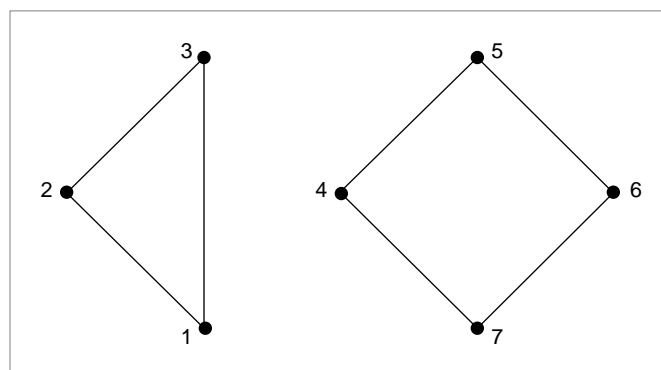


However, if the sample function had specified the `gxBreakLeftEdit` constant, as with the call

```
GXSetPolygonParts(aPolygonsShape, 4, 0,
                  (gxPolygons *) newCenterGeometry,
                  gxBreakLeftEdit);
```

QuickDraw GX would break the resulting polygon into two contours: The `gxBreakLeftEdit` constant indicates that the polygon should be broken between the newly inserted point and the previous point, as shown in Figure 2-57.

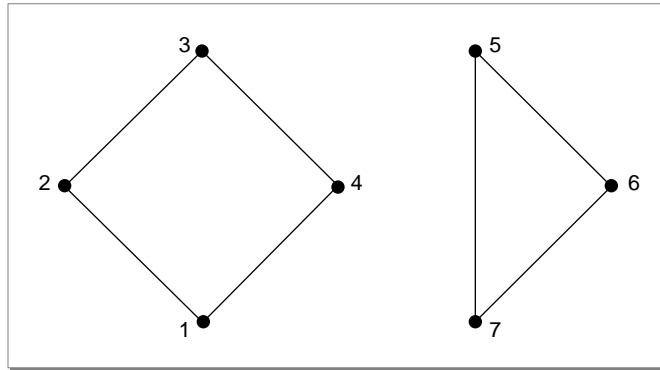
Figure 2-57 A polygon shape edited with the `gxBreakLeftEdit` flag set



Geometric Shapes

The `gxBreakRightEdit` constant works similarly, except the break occurs between the newly inserted point and the subsequent point, as shown in Figure 2-58.

Figure 2-58 A polygon shape edited with the `gxBreakRightEdit` flag set



You can use the `GXSetPolygonParts` function to insert a polygon geometry that contains multiple contours. In this case, the breaks that occur in the inserted geometry remain in the resulting polygon shape.

For more information about polygon geometries, see “Polygon Shapes” on page 2-22.

For more information about the `GXGetPolygonParts` and `GXSetPolygonParts` functions, see the function descriptions on page 2-144 and page 2-145.

The next two sections show more examples of editing shape geometries.

Editing Paths Parts

The `GXGetPathParts` and `GXSetPathParts` functions work similarly to the `GXGetPolygonParts` and `GXSetPolygonParts` functions, which are described in the previous section.

The sample function in Listing 2-31 creates a path shape similar to the polygon shape from the previous section. Later examples in this section use this path shape to demonstrate editing path parts.

Listing 2-31 Creating a path shape with two curved contours

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

    static long twoCurveGeometry[] = {2, /* number of contours */
                                       3, /* number of points */
                                       0x40000000, /* 0100 ... */
                                       ff(100), ff(150), /* on */
                                       ff(50), ff(100), /* off */
                                       ff(100), ff(50), /* on */
                                       3, /* number of points */
                                       0x40000000, /* 0100 ... */
                                       ff(200), ff(50), /* on */
                                       ff(250), ff(100), /* off */
                                       ff(200), ff(150)}; /* on */

    aPathShape = GXNewPaths((gxPaths *) twoCurveGeometry);
    GXSetShapeFill(aPathShape, gxOpenFrameFill);

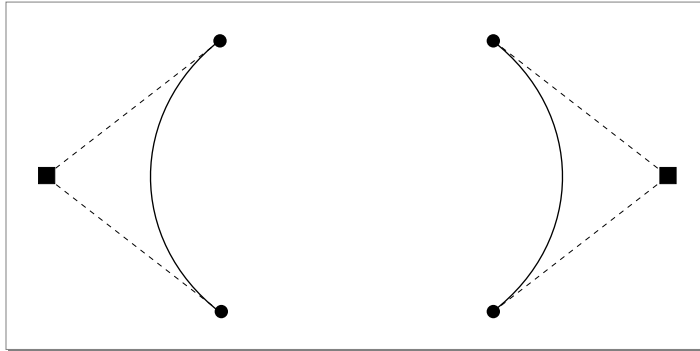
    GXDrawShape(aPathShape);

    GXDisposeShape(aPathShape);
}
```

Geometric Shapes

The resulting path shape is shown in Figure 2-59.

Figure 2-59 A path shape with two curved contours



You can use the `GXSetPathParts` function to replace any number of geometric points from this path shape with an arbitrary number of new geometric points. In a manner similar to the `GXSetPolygonParts` function, the `GXSetPathParts` function requires that you encapsulate the new geometric points in a path geometry. For example, to replace the top two geometric points in the path shape shown in Figure 2-59 with a single geometric point, you must first encapsulate the new geometric point in a path geometry, as with the definition

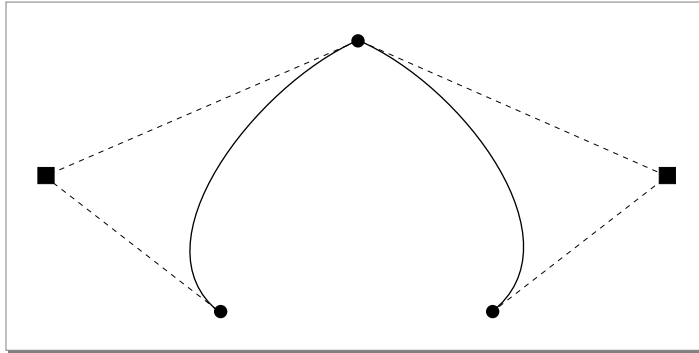
```
static long newTopGeometry[] = {1, /* number of contours */
                                1, /* number of points */
                                0x00000000, /* 0000 ... */
                                ff(150), ff(50)}; /* on curve */
```

and then call the `GXSetPathParts` function:

```
GXSetPathParts(aPathsShape, 3, 2,
               (gxPaths *) newTopGeometry, gxBreakNeitherEdit);
```


The resulting path shape is shown in Figure 2-60.

Figure 2-60 A path shape edited with `GXSetPathParts`



For more information about path geometries, see “Path Shapes” beginning on page 2-25.

For more information about the `GXGetPathParts` and `GXSetPathParts` functions, see the function descriptions on page 2-148 and page 2-149.

Editing Shape Parts

The `GXSetShapeParts` function is more general than the `GXSetPolygonParts` and `GXSetPathParts` functions described in the previous two sections. The `GXSetShapeParts` function allows you to replace a subset of the geometric points in one shape with the geometric points in the geometry of another shape.

For example, with `GXSetShapeParts` you could replace the last three geometric points of a polygon shape with the geometry of a line shape, or you could replace the first geometric point of a path shape with the entire geometry of a polygon shape.

Geometric Shapes

The sample function in Listing 2-32 creates a path shape with one contour. Later examples in this section use this path shape to demonstrate editing shape parts.

Listing 2-32 Creating a path shape with one contour

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

    static long twoCurveGeometry[] = {1, /* number of contours */
                                       6, /* number of points */
                                       0x48000000, /* 0100 1000 */
                                       ff(100), ff(150), /* on */
                                       ff(50),  ff(100), /* off */
                                       ff(100), ff(50),  /* on */
                                       ff(200), ff(50),  /* on */
                                       ff(250), ff(100), /* off */
                                       ff(200), ff(150)}; /* on */

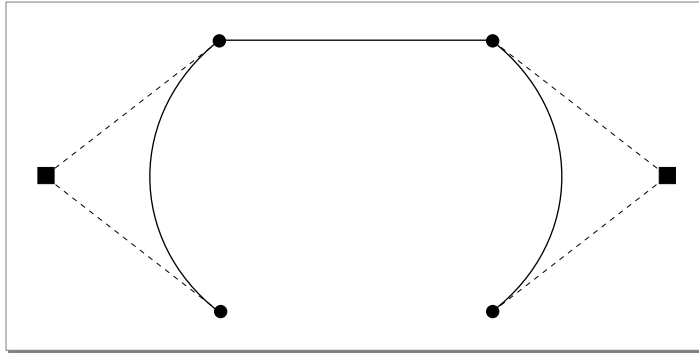
    aPathsShape = GXNewPaths((gxPaths *) twoCurveGeometry);
    GXSetShapeFill(aPathShape, gxOpenFrameFill);

    GXDrawShape(aPathShape);

    GXDisposeShape(aPathShape);
}
```

The resulting shape is shown in Figure 2-61.

Figure 2-61 A path shape with a flat top



To insert a new geometric point in this shape using the `GXSetShapeParts` function, you must first encapsulate the new geometric point in a point shape:

```
static gxPoint newTopGeometry = {ff(150), ff(20)};
gxShape aPointShape;
```

```
aPointShape = GXNewPoint(&newTopGeometry);
```

Then you call the `GXSetShapeParts` function:

```
GXSetShapeParts(aPathsShape, 4, 0, aPointShape,
                gxBreakNeitherEdit);
```

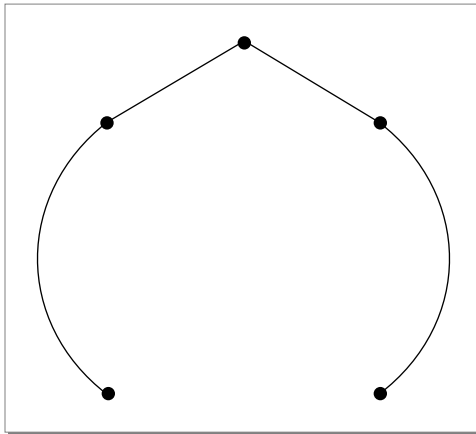
Geometric Shapes

Since you must create a shape to encapsulate the point geometry, you should dispose of this shape when you no longer need it:

```
GXDisposeShape(aPointShape);
```

The resulting path shape is shown in Figure 2-62.

Figure 2-62 A path shape edited to have a pointy top



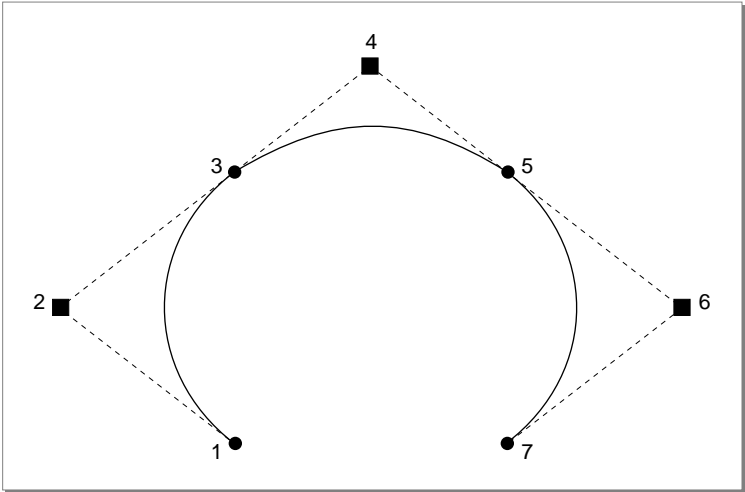
You can also use the `GXSetShapeParts` function to insert an off-curve control point in the path shape. To do this, however, you must encapsulate the new geometric point into a path shape, because only a path shape can contain a single off-curve point.

```
gxShape aSingleOffCurvePoint;
static long newTopGeometry[] = {1, /* number of contours */
                                1, /* number of points */
                                0x80000000, /* 1000 ... */
                                ff(150), ff(20)}; /* off curve */

aSingleOffCurvePoint = GXNewPaths((gxPaths *) newTopGeometry);
GXSetShapeParts(aPathsShape, 4, 0,
                aSingleOffCurvePoint, gxBreakNeitherEdit);
GXDisposeShape(aSingleOffCurvePoint);
```

The resulting path shape is shown in Figure 2-63.

Figure 2-63 A path shape edited to have a round top



Geometric Shapes

The `GXSetShapeParts` function allows you to edit the geometry of any shape. For example, the sample function in Listing 2-33 creates a line shape and uses `GXSetShapeParts` to change the last point.

Listing 2-33 Creating a diagonal line

```
void CreateDiagonalLine(void)
{
    gxShape aLineShape;
    gxShape aPointShape;

    static gxLine lineGeometry = {ff(50), ff(50),
                                   ff(150), ff(150)};

    static gxPoint newLastPointGeometry = {ff(300), ff(150)};

    aLineShape = GXNewLine(&lineGeometry);
    GXSetShapeFill(aLineShape, gxOpenFrameFill);

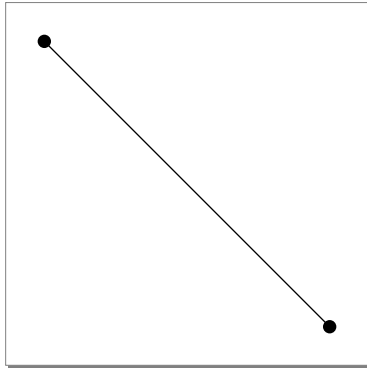
    aPointShape = GXNewPoint(&newLastPointGeometry);
    GXSetShapeParts(aLineShape, 2, 1, aPointShape,
                    gxBreakNeitherEdit);
    GXDisposeShape(aPointShape);

    GXDrawShape(aLineShape);

    GXDisposeShape(aLineShape);
}
```

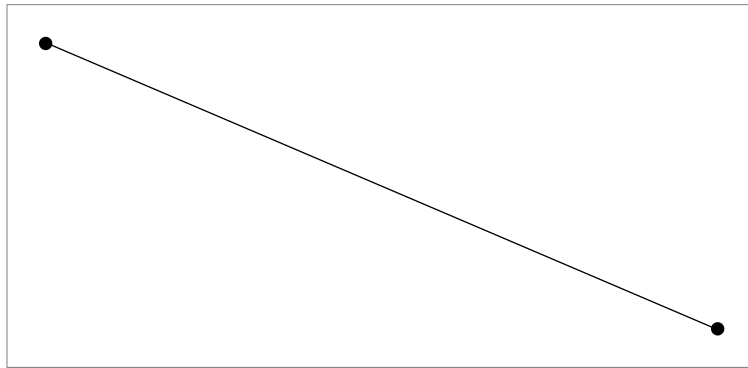
The original line is shown in Figure 2-64.

Figure 2-64 A diagonal line



The line shape with the replaced last point is shown in Figure 2-65.

Figure 2-65 An edited line



For more information about editing shape parts and the `GXSetShapeParts` function, see the function description on page 2-154.

Applying Functions Described Elsewhere to Geometric Shapes

QuickDraw GX provides many functions that apply exclusively to geometric shapes. However, there are many other QuickDraw GX functions that apply to other types of shapes as well as geometric shapes.

The next two sections discuss how functions described elsewhere operate on geometric shapes. These sections are:

- “Shape-Related Functions Applicable to Geometric Shapes,” the next section
- “Other Functions Applicable to Geometric Shapes,” on page 2-103

Shape-Related Functions Applicable to Geometric Shapes

You can apply all of the functions described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects* to geometric shapes. These functions allow you to

- manipulate the shape object that represents geometric shapes (for example, you can copy, clone, cache, compare, and dispose of a geometric shape)
- set the geometry, shape type, shape fill, and shape attributes of geometric shapes
- change the style, ink, and transform objects that are associated with geometric shapes
- manipulate the tags and owner count of the geometric shapes

Table 2-1 gives important information about geometric shapes for a subset of the functions from the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*. Functions described in that chapter that do not appear in this list exhibit the same behavior when applied to geometric shapes as they do when applied to other types of shapes.

Table 2-1 Shape-related functions that exhibit special behavior with geometric shapes

Function name	Action taken
GXGetDefaultShape	Returns a reference to the default geometric shape of the specified type. See “The Geometric Shape Types” beginning on page 2-16 for information about the default geometric shapes.
GXGetShapeFill	Returns the shape fill of the shape. See “The Geometric Shape Types” beginning on page 2-16 for a discussion of which shape fills are appropriate for which geometric shapes.
GXSetDefaultShape	Allows you to specify the shape to copy when creating new geometric shapes. See “The Geometric Shape Types” beginning on page 2-16 for information about the default geometric shapes.
GXSetShapeFill	Sets the shape fill of the shape. See “The Geometric Shape Types” beginning on page 2-16 for a discussion of which shape fills are appropriate for which geometric shapes.
GXSetShapeType	Changes the shape type of the geometric shape and converts the shape fill and geometry as appropriate. See the rest of this section for more information about converting shape types.

When converting between geometric shape types, the behavior of the GXSetShapeType function depends on the new shape type. If the new shape type is the point, line or rectangle type, the new geometry is based on the bounding rectangle of the original geometry:

Old type	New type	New geometry
Any	Point	Upper-left corner of bounds
Any	Line	Line from upper-left corner to lower-left corner
Any	Rectangle	Bounding rectangle of original geometry

For examples, see “Converting Between Geometric Shape Types” beginning on page 2-65.

Geometric Shapes

If the new shape type is the curve type, the conversion performed depends on the original shape type:

Old type	New type	New geometry
Point	Curve	New control points all set to original point
Line	Curve	First and last points remain the same; off-curve control point set equal to last point
Rectangle	Curve	First point set to original upper-left point; last point set to original lower-right point; off-curve control point set equal to last point
Polygon	Curve	New control points set to first three original control points
Path	Curve	New control points set to first three original control points

For examples, see “Converting Shapes to Curve Shapes” beginning on page 2-71.

If the new shape type is the polygon type, this function retains all of the original geometric points:

Old type	New type	New geometry
Point, line, or rectangle	Polygon	Single contour with same geometric points
Curve	Polygon	Single contour with same geometric points; the off-curve point becomes on curve
Path	Polygon	Same geometric points; all on curve (calculates approximation if curve error is not zero)

When converting a path shape to a polygon shape, this function examines the curve error of the style of the path shape. If the curve error is not zero, this function creates a polygon approximation of the original path. For more information about curve error, see the next chapter, “Geometric Styles,” in this book.

Finally, if the new shape type is the path type, the `GXSetShapeType` function retains all of the original geometry information:

Old type	New type	New geometry
Point, line, curve, or rectangle	Path	Single contour with same geometric points
Polygon	Path	Same number of contours; same geometric points; all control points remain on curve

For examples, see “Converting Shapes to Polygons and Paths” beginning on page 2-74.

Other Functions Applicable to Geometric Shapes

You can apply any of the geometric operations described in Chapter 4, “Geometric Operations,” to the geometric shapes.

Geometric shapes make use of the geometric properties of their style objects. For this reason, you may apply shape-based functions (such as `GXSetShapePen`, `GXSetShapeDash`, and so on) described in Chapter 3, “Geometric Styles,” to geometric shapes.

You may also apply any of the shape-based functions in the chapter “Typographic Styles” in *Inside Macintosh: QuickDraw GX Typography* to geometric shapes. However, these functions do not affect the way geometric shapes appear when drawn.

You may apply any of the shape-based functions described in the chapter “Ink Objects” in *Inside Macintosh: QuickDraw GX Typography* to geometric shapes.

These functions include `GXSetShapeColor`, `GXSetShapeTransfer`, `GXSetShapeInkAttributes`, and so on.

You may apply any of the shape-based functions described in the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Typography* to geometric shapes. These functions include `GXSetShapeClip`, `GXSetShapeMapping`, `GXSetShapeHitTest`, and so on.

Geometric Shapes Reference

This section describes the data types and functions that are related to geometric shapes.

The “Data Types” section shows the structure definitions for the geometries of the geometric shapes.

The “Functions” section describes the functions that allow you to create and draw geometric shapes and functions that allow you to perform simple manipulations on shape geometries, such as replacing the entire geometry, or replacing certain points in a geometry.

Chapter 4, “Geometric Operations,” in this book describes functions that allow you to perform more advanced operations on shape geometries—operations such as inseting, intersecting, and so on.

Data Types

This section describes the structures that you use when creating and manipulating geometric shapes.

You use the `gxPoint` structure when creating a point shape and when specifying geometric point positions for all of the geometric shapes.

You use the `gxLine` structure when creating a line shape.

You use the `gxCurve` structure when creating a curve shape.

You use the `gxRectangle` structure when creating a rectangle shape and when specifying the bounding rectangle of a shape.

You use the `gxPolygon` structure when specifying a single contour made up of straight lines. You use the `gxPolygons` structure when specifying multiple contours made up of straight lines.

You use the `gxPath` structure when specifying a single contour made up of straight lines and curves. You use the `gxPaths` structure when specifying multiple path contours.

The Point Structure

You use the `gxPoint` structure in a number of situations; for example, to specify the geometry of a point shape, to specify the position of geometric points in the geometries of other geometric shape types, to specify a location to hit-test, to specify the position of a bitmap, and so on.

The `gxPoint` structure is defined as follows:

```
struct gxPoint {
    Fixed    x;
    Fixed    y;
};
```

Field descriptions

<code>x</code>	A horizontal distance. Greater values of the <code>x</code> field indicate distances further to the right.
<code>y</code>	A vertical distance. Greater values of the <code>y</code> field indicate distances further down.

The location of the origin depends on the context where you use the point; for example, it might be the upper-left corner of a view port.

Notice that the `x` and `y` fields are of type `Fixed`. QuickDraw GX allows you to specify fractional coordinate positions.

For more information about coordinates and coordinate spaces, see *Inside Macintosh: QuickDraw GX Objects*.

For more information about points and point shapes, see “Point Shapes” on page 2-16.

The Line Structure

You use the `gxLine` structure to specify the geometry of a line shape.

The `gxLine` structure is defined as follows:

```
struct gxLine {  
    struct gxPoint first;  
    struct gxPoint last;  
};
```

Field descriptions

<code>first</code>	The coordinate position where the line begins.
<code>last</code>	The coordinate position where the line ends.

Notice that the endpoints of a line are ordered—lines have an implicit direction. This direction can affect how QuickDraw GX draws a line shape, particularly when the line shape has stylistic variations.

For more information about lines and line shapes, see “Line Shapes” on page 2-17.

The Curve Structure

You use the `gxCurve` structure to specify the geometry of a curve shape.

The `gxCurve` structure is defined as follows:

```
struct gxCurve {  
    struct gxPoint first;  
    struct gxPoint control;  
    struct gxPoint last;  
};
```

Field descriptions

<code>first</code>	The coordinate position where the curve begins.
<code>control</code>	The coordinate position of the off-curve control point, which QuickDraw GX uses to determine the tangents of the curve.
<code>last</code>	The coordinate position where the curve ends.

The curve defined by these three points is a quadratic Bézier curve.

Because the geometric points that define a curve are ordered, curves have direction. The direction of a curve can affect how QuickDraw GX draws the curve shape, particularly when the curve shape has stylistic variations.

For more information about curves and curve shapes, see “Curve Shapes” on page 2-18.

The Rectangle Structure

You use the `gxRectangle` structure in a variety of situations: to specify the geometry of a rectangle shape, to specify the bounding rectangle of another shape, and so on.

The `gxRectangle` structure is defined as follows:

```
struct gxRectangle {
    Fixed    left;
    Fixed    top;
    Fixed    right;
    Fixed    bottom;
};
```

Field descriptions

<code>left</code>	Specifies the x-coordinate of the rectangle's first geometric point.
<code>top</code>	Specifies the y-coordinate of the rectangle's first geometric point.
<code>right</code>	Specifies the x-coordinate of the rectangle's last geometric point.
<code>bottom</code>	Specifies the y-coordinate of the rectangle's last geometric point.

You may specify a rectangle's geometric points in any order—the coordinates in the `left` and `top` field do not have to correspond to the rectangle's upper-left corner. However, rectangles calculated by QuickDraw GX, such as those returned from geometric operations as described in Chapter 4, "Geometric Operations," always have their coordinates specified in the standard order.

For more information about rectangles and rectangle shapes, see "Rectangle Shapes" on page 2-20.

Polygon Structures

You use the `gxPolygon` structure to specify a single polygon contour composed of straight lines.

The `gxPolygon` structure is defined as follows:

```
struct gxPolygon {
    long          vectors;
    struct gxPoint vector[gxAnyNumber];
};
```

Field descriptions

<code>vectors</code>	The number of geometric points in the contour.
<code>vector</code>	The coordinates of the geometric points.

The array index `gxAnyNumber` indicates that the `gxPolygon` data structure is a variable-length structure—it can include any number of points.

Geometric Shapes

The `gxPolygons` structure allows you to group multiple polygon contours together. You use this structure when specifying the geometry of a polygon shape.

The `gxPolygons` structure is defined as follows:

```
struct gxPolygons {
    long          contours;
    struct gxPolygon contour[gxAnyNumber];
};
```

Field descriptions

`contours` The number of polygon contours.

`contour` The polygon contours.

The array index `gxAnyNumber` indicates that the `gxPolygons` data structure is also a variable-length structure—it can include any number of `gxPolygon` structures.

Implementation Note

In version 1.0 of QuickDraw GX, a single polygon contour can have between 1 and 32,767 geometric points. The geometry of a polygon shape can have between 0 and 32,767 polygon contours. The total size of a polygon geometry may not exceed 2,147,483,647 bytes. ♦

For more information about polygons and polygon shapes, see “Polygon Shapes” on page 2-22.

Path Structures

You use the `gxPath` structure to specify a single contour composed of straight lines and curves.

The `gxPath` structure is defined as follows:

```
struct gxPath {
    long          vectors;
    long          controlBits[gxAnyNumber];
    struct gxPoint vector[gxAnyNumber];
};
```

Field descriptions

`vectors` The number of geometric points in the contour.

`controlBits` Bit flags that indicate which geometric points are on curve and which are off-curve control points.

`vector` The coordinates of the geometric points.

The array index `gxAnyNumber` indicates that the `gxPath` data structure is a variable-length structure—it can include any number of geometric points

Geometric Shapes

Each bit in the array specified in the `controlBits` field indicates whether a particular point in the array specified by the vector field is on curve or off curve. A value of 0 indicates that the corresponding point is on curve and a value of 1 indicates that the corresponding point is off curve.

The `gxPaths` structure allows you to group multiple path contours together. You use this data structure when specifying the geometry of a path shape.

The `gxPaths` structure is defined as follows:

```
struct gxPaths {
    long          contours;
    struct gxPath contour[gxAnyNumber];
};
```

Field descriptions

`contours` The number of path contours.
`contour` The path contours.

The array index `gxAnyNumber` indicates that the `gxPaths` data structure is also a variable-length structure—it can include any number of path contours.

Implementation Note

In version 1.0 of QuickDraw GX, a single path contour can have between 0 and 32,767 geometric points. The geometry of a path shape can between 0 and 32,767 polygon contours. The total size of a path geometry may not exceed 2,147,483,647 bytes. ♦

For more information about paths and path shapes, see “Path Shapes” on page 2-25.

Functions

This section describes the functions available for

- creating new geometric shapes
- manipulating the geometries of geometric shapes
- editing parts of shape geometries
- drawing geometric shapes

Chapter 4, “Geometric Operations,” contains information about more sophisticated functions for manipulating shape geometries.

Geometric Shapes

For information about creating, drawing, and manipulating bitmap shapes, see Chapter 5, “Bitmap Shapes.”

For information about creating, drawing, and manipulating picture shapes, see Chapter 6, “Picture Shapes.”

For information about getting and setting the default geometric shapes and information about manipulating shape type and shape fill, see the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*. For information about hit-testing geometric shapes, see the chapter “Transform Objects” also in that book.

For information about creating, drawing, and manipulating typographic shapes, see *Inside Macintosh: QuickDraw GX Typography*.

Creating Geometric Shapes

QuickDraw GX provides a number of ways for you to create a new shape.

The functions described in this section allow you to specify a shape’s initial geometry when creating the shape. For example, the `GXNewShapeVector` function allows you to specify a shape type and an array of values. The function creates a new shape of the specified type and uses the array of values to initialize the new shape’s geometry.

The `GXNewPoint`, `GXNewLine`, `GXNewCurve`, `GXNewRectangle`, `GXNewPolygons`, and `GXNewPaths` functions all create a new shape of a specific type. These functions allow you to specify the shape’s initial geometry.

You can also use the `GXNewShape` function to create shapes. This function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*, allows you to create a shape by specifying only the shape type; the geometry of the new shape is set to its initial state—all geometric points are (0.0, 0.0) and polygons and paths have 0 contours. You can customize the shape’s geometry using the functions described in “Getting and Setting Shape Geometries” beginning on page 2-119.

GXNewShapeVector

You can use the `GXNewShapeVector` function to create a new shape of any type.

```
void GXNewShapeVector(gxShapeType aType, const Fixed vector[]);
```

`aType` A reference to the shape whose geometry you want to change.

`vector` An array of fixed-point values to use as the new geometry.

function result A reference to the new shape.

Geometric Shapes

DESCRIPTION

The `GXNewShapeVector` function copies the default shape of the shape type specified by the `aType` parameter, sets the owner count of the new shape to 1, initializes its geometry with the values in the `vector` parameter, and returns a reference to it as the function result.

Although this function creates a copy of the default shape, it does not create a copy of the default shape's style, ink, or transform. The new shape returned by this function contains references to same style, ink, and transform as the default shape. You can change the style using functions from Chapter 3, "Geometric Styles," and you can change the style, ink, and transform using functions from *Inside Macintosh: QuickDraw GX Objects*.

You may pass any number of values in the `vector` array; the `GXNewShapeVector` function traverses this array as necessary to initialize the new shape's geometry. If you pass too few values in this parameter, the function posts the warning `extra_data_passed_was_ignored`.

If you specify a shape type that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Creates a bitmap shape; expects the vector array to contain values corresponding to the fields of a bitmap structure
picture	Creates a picture shape with no overriding styles, inks, or transforms; expects the vector array to contain an array of shape references
text	Posts the error <code>graphic_type_does_not_contain_points</code>
glyph	Posts the error <code>graphic_type_does_not_contain_points</code>
layout	Posts the error <code>graphic_type_does_not_contain_points</code>

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>illegal_type_for_shape</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>graphic_type_does_not_contain_points</code>	(debugging version)

Warnings

`extra_data_passed_was_ignored`

SEE ALSO

For general information about each type of geometry, see “About Geometric Shapes” on page 2-5. For specific definitions of each type of geometry, see the section “Data Types” beginning on page 2-104.

For information about related functions, see the descriptions of the `GXNewPoint`, `GXNewLine`, `GXNewCurve`, `GXNewRectangle`, `GXNewPolygons`, and `GXNewPaths` functions on page 2-111 through page 2-119.

GXNewPoint

You can use the `GXNewPoint` function to create a new point shape and initialize its geometry.

```
gxShape GXNewPoint(const gxPoint *data);
```

`data` A pointer to the initial point geometry.

function result A reference to the new point shape.

DESCRIPTION

The `GXNewPoint` function creates a copy of the default point shape, sets the owner count of the copy to 1, initializes its geometry with the values in the `data` parameter, and returns a reference to it as the function result.

Although this function creates a copy of the default point shape, it does not create a copy of the default point’s style, ink, or transform objects. The new point shape returned by this function contains references to the same style, ink, and transform as the default point shape.

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewPoint` 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 creating and disposing of shapes.

If an error occurs, this function returns `nil` as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`parameter_is_nil` (debugging version)

Geometric Shapes

SEE ALSO

For an example that uses this function, see “Creating and Drawing Points” beginning on page 2-29.

For a discussion of points and the default point shape, see “Point Shapes” on page 2-16.

For a description of the `gxPoint` structure, see page 2-104.

To create a new point shape without specifying an initial geometry, see the description of the `GXNewShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To set the geometry of an existing point shape, see the description of the `GXSetPoint` function on page 2-122.

To draw a point geometry, see the description of `GXDrawPoint` on page 2-158. To draw a point shape, see the description of `GXDrawShape` in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXNewLine

You can use the `GXNewLine` function to create a new line shape and initialize its geometry.

```
gxShape GXNewLine(const gxLine *data);
```

`data` A pointer to the initial line geometry.

function result A reference to the new line shape.

DESCRIPTION

The `GXNewLine` function creates a copy of the default line shape, sets the owner count of the copy to 1, initializes its geometry with the values in the `data` parameter, and returns a reference to it as the function result.

Although this function creates a copy of the default line shape, it does not create a copy of the default line’s style, ink, or transform objects. The new line shape returned by this function contains references to same style, ink, and transform as the default line shape.

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewLine` 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 creating and disposing of shapes.

If an error occurs, this function returns `nil` as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`parameter_is_nil` (debugging version)

SEE ALSO

For an example that uses this function, see “Creating and Drawing Lines” beginning on page 2-36.

For a discussion of lines and the default line shape, see “Line Shapes” on page 2-17.

For a description of the `gxLine` structure, see page 2-105.

To create a new line shape without specifying an initial geometry, see the description of the `GXNewShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To set the geometry of an existing line shape, see the description of the `GXSetLine` function on page 2-124.

To draw a line geometry without creating a line shape, see the description of `GXDrawLine` on page 2-158. To draw a line shape, see the description of `GXDrawShape` in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXNewCurve

You can use the `GXNewCurve` function to create a new curve shape and initialize its geometry.

```
gxShape GXNewCurve(const gxCurve *data);
```

`data` A pointer to the initial curve geometry.

function result A reference to the new curve shape.

DESCRIPTION

The `GXNewCurve` function creates a copy of the default curve shape, sets the owner count of the copy to 1, initializes its geometry with the values in the `data` parameter, and returns a reference to it as the function result.

Although this function creates a copy of the default curve shape, it does not create a copy of the default curve’s style, ink, or transform objects. The new curve shape returned by this function contains references to same style, ink, and transform as the default curve shape.

Geometric Shapes

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewCurve` 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 creating and disposing of shapes.

If an error occurs, this function returns `nil` as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`parameter_is_nil` (debugging version)

SEE ALSO

For an example that uses this function, see “Creating and Drawing Curves” beginning on page 2-41.

For a discussion of curves and the default curve shape, see “Curve Shapes” beginning on page 2-18.

For a description of the `gxCurve` structure, see page 2-105.

To create a new curve shape without specifying an initial geometry, see the description of the `GXNewShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To set the geometry of an existing curve shape, see the description of the `GXSetCurve` function on page 2-126.

To draw a curve geometry without creating a curve shape, see the description of `GXDrawCurve` on page 2-159. To draw a curve shape, see the description of `GXDrawShape` in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXNewRectangle

You can use the `GXNewRectangle` function to create a new rectangle shape and initialize its geometry.

```
gxShape GXNewRectangle(const gxRectangle *data);
```

`data` A pointer to the initial rectangle geometry.

function result A reference to the new rectangle shape.

Geometric Shapes

DESCRIPTION

The `GXNewRectangle` function creates a copy of the default rectangle shape, sets the owner count of the copy to 1, initializes its geometry with the values in the `data` parameter, and returns a reference to it as the function result.

Although this function creates a copy of the default rectangle shape, it does not create a copy of the default rectangle's style, ink, or transform objects. The new rectangle shape returned by this function contains references to same style, ink, and transform as the default rectangle shape.

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewRectangle` 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 creating and disposing of shapes.

If an error occurs, this function returns `nil` as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`parameter_is_nil` (debugging version)

SEE ALSO

For an example that uses this function, see "Creating and Drawing Rectangles" beginning on page 2-43.

For a discussion of rectangles and the default rectangle shape, see "Rectangle Shapes" beginning on page 2-20.

For a description of the `gxRectangle` structure, see page 2-106.

To create a new rectangle shape without specifying an initial geometry, see the description of the `GXNewShape` function in the chapter "Shape Objects" in *Inside Macintosh: QuickDraw GX Objects*.

To set the geometry of an existing rectangle shape, see the description of the `GXSetRectangle` function on page 2-129.

To draw a rectangle geometry without creating a rectangle shape, see the description of `GXDrawRectangle` on page 2-160. To draw a rectangle shape, see the description of `GXDrawShape` in the chapter "Shape Objects" in *Inside Macintosh: QuickDraw GX Objects*.

GXNewPolygons

You can use the `GXNewPolygons` function to create a new polygon shape and initialize its geometry.

```
gxShape GXNewPolygons(const gxPolygons *data);
```

`data` A pointer to the initial polygon geometry.

function result A reference to the new polygon shape.

DESCRIPTION

The `GXNewPolygons` function creates a copy of the default polygon shape, sets the owner count of the copy to 1, initializes its geometry with the values in the `data` parameter, and returns a reference to it as the function result. If you specify `nil` for the `data` parameter, this function returns a polygon shape with no polygon contours.

Although this function creates a copy of the default polygon shape, it does not create a copy of the default polygon's style, ink, or transform objects. The new polygon shape returned by this function contains references to same style, ink, and transform as the default polygon shape.

Implementation Note

In version 1.0 of QuickDraw GX, the total size of a polygon geometry may not exceed 2,147,483,647 bytes. If the size of the data you provide in the `data` parameter exceeds this limit, the `GXNewPolygons` function posts a `size_of_polygon_exceeds_implementation_limit` error. ♦

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewPolygons` 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 creating and disposing of shapes.

If an error occurs, this function returns `nil` as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

```

out_of_memory
number_of_points_exceeds_implementation_limit
number_of_contours_exceeds_implementation_limit
size_of_polygon_exceeds_implementation_limit
count_is_less_than_one

```

(debugging version)

SEE ALSO

For an example that uses this function, see “Creating and Drawing Polygons” beginning on page 2-45.

For a discussion of polygons and the default polygon shape, see “Polygon Shapes” beginning on page 2-22.

For a description of the `gxPolygons` structure, see page 2-106.

To create a new polygon shape without specifying an initial geometry, see the description of the `GXNewShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To set the geometry of an existing polygon shape, see the description of the `GXSetPolygons` function on page 2-131.

To draw a polygon geometry without creating a polygon shape, see the description of `GXDrawPolygons` on page 2-161. To draw a polygon shape, see the description of `GXDrawShape` in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXNewPaths

You can use the `GXNewPaths` function to create a new path shape and initialize its geometry.

```
gxShape GXNewPaths(const gxPaths *data);
```

`data` A pointer to the initial path geometry.

function result A reference to the new path shape.

Geometric Shapes

DESCRIPTION

The `GXNewPaths` function creates a copy of the default path shape, sets the owner count of the copy to 1, initializes its geometry with the values in the `data` parameter, and returns a reference to it as the function result. If you specify `nil` for the `data` parameter, this function returns a path shape with no path contours.

Although this function creates a copy of the default path shape, it does not create a copy of the default path shape's style, ink, or transform objects. The new path shape returned by this function contains references to same style, ink, and transform as the default path shape.

Implementation Limit

In version 1.0 of QuickDraw GX, the total size of a path geometry may not exceed 2,147,483,647 bytes. If the size of the data you provide in the `data` parameter exceeds this limit, the `GXNewPaths` function posts a `size_of_path_exceeds_implementation_limit` error. ♦

SPECIAL CONSIDERATIONS

If no error occurs, the `GXNewPaths` 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 creating and disposing of shapes.

If an error occurs, this function returns `nil` as the function result.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>count_is_less_than_one</code>	(debugging version)

SEE ALSO

For an example that uses this function, see “Creating and Drawing Paths” beginning on page 2-55.

For a discussion of paths and the default path shape, see “Path Shapes” beginning on page 2-25.

For a description of the `gxPaths` structure, see page 2-107.

To create a new path shape without specifying an initial geometry, see the description of the `GXNewShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

To set the geometry of an existing path shape, see the description of the `GXSetPaths` function on page 2-133.

Geometric Shapes

To draw a path geometry without creating a path shape, see the description of `GXDrawPaths` on page 2-162. To draw a path shape, see the description of `GXDrawShape` in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Getting and Setting Shape Geometries

The geometry property of geometric shapes contains the geometric points that define the shape. The geometries of polygon shapes and path shapes also contain some additional information, such as the number of separate contours, how many geometric points in each contour, and (for paths) which geometric points are on curve and which are off-curve control points.

For general information about each type of geometry, see “About Geometric Shapes” beginning on page 2-5. For specific definitions of each type of geometric structure, see the section “Data Types” beginning on page 2-104.

The `GXSetShapeVector` function allows you to change the geometry of any shape. With this function, you specify a shape and an array of values. The function replaces the geometry of the specified shape with the values in the array. This function works for other shape types as well as geometric shapes.

The `GXGetPoint`, `GXGetLine`, `GXGetCurve`, `GXGetRectangle`, `GXGetPolygons`, and `GXGetPaths` functions each return the geometry of a specific type of shape.

The `GXSetPoint`, `GXSetLine`, `GXSetCurve`, `GXSetRectangle`, `GXSetPolygons`, and `GXSetPaths` functions each replace the geometry of a specific type of shape.

GXSetShapeVector

You can use the `GXSetShapeVector` function to change the geometry of an existing shape.

```
void GXSetShapeVector(gxShape target, const Fixed vector[]);
```

target A reference to the shape whose geometry you want to change.

data An array of fixed-point values to use as the new geometry.

DESCRIPTION

The `GXSetShapeVector` function replaces the geometry of the `target` shape with a new geometry, which it creates by traversing the vector array. The length of the vector array that you supply depends on shape type of the `target` shape; for example, if the `target` shape is a point, you should provide a vector array with two `Fixed` values; if the `target` shape is a line, you should provide four `Fixed` values, and so on.

Geometric Shapes

Although this function creates a copy of the default shape, it does not create a copy of the default shape's style, ink, or transform. The new shape returned by this function contains references to same style, ink, and transform as the default shape. You can change the style using functions from Chapter 3, "Geometric Styles," and you can change the style, ink, and transform using functions from *Inside Macintosh: QuickDraw GX Objects*.

You may pass any number of values in the vector array; the `GXNewShapeVector` function traverses this array as necessary to initialize the new shape's geometry. If you pass too few values in this parameter, the function posts the warning `extra_data_passed_was_ignored`.

If you specify a shape type that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Sets the target shape to be a bitmap shape; expects the vector array to contain values corresponding to the fields of a bitmap structure
picture	Sets the target shape to be a picture shape with no overriding styles, inks, or transforms; expects the vector array to contain an array of shape references
text	Posts the error <code>graphic_type_does_not_contain_points</code>
glyph	Posts the error <code>graphic_type_does_not_contain_points</code>
layout	Posts the error <code>graphic_type_does_not_contain_points</code>

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>illegal_type_for_shape</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>graphic_type_does_not_contain_points</code>	(debugging version)

Warnings

<code>extra_data_passed_was_ignored</code>
--

SEE ALSO

For general information about each type of geometry, see “About Geometric Shapes” on page 2-5. For specific definitions of each type of geometry, see the section “Data Types” beginning on page 2-104.

For information about related functions, see the descriptions of the `GXSetPoint`, `GXSetLine`, `GXSetCurve`, `GXSetRectangle`, `GXSetPolygons`, and `GXSetPaths` functions on page 2-122 through page 2-135.

GXGetPoint

You can use the `GXGetPoint` function to determine the geometry of an existing point shape.

```
gxPoint *GXGetPoint(gxShape source, gxPoint *data);
```

source A reference to the point shape whose geometry you want to determine.
data A pointer to a `gxPoint` structure. The function copies the source shape’s geometry into this structure.

function result A pointer to a copy of the source shape’s geometry.

DESCRIPTION

The `GXGetPoint` function copies the geometry information from the source point shape into the `gxPoint` structure pointed to by the `data` parameter. As a convenience, this function also returns a pointer to the point geometry as the function result.

If the source shape is not a point shape, this function posts the error code `illegal_type_for_shape`.

You must pass a pointer to a `gxPoint` structure in the `data` parameter—if you pass `nil` for this parameter, the function posts the error code `parameter_is_nil`.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>illegal_type_for_shape</code>	(debugging version)
<code>parameter_is_nil</code>	(debugging version)

Geometric Shapes

SEE ALSO

For general information about point geometries, see “Point Shapes” on page 2-16.

For the definition of the `gxPoint` structure, see page 2-104.

To create a new point shape, use the `GXNewPoint` function, which is described on page 2-111.

To change the geometry of an existing point shape, use the `GXSetPoint` function, which is described in the next section.

To draw a point geometry without creating a point shape, use the `GXDrawPoint` function, which is described on page 2-158. To draw a point shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXSetPoint

You can use the `GXSetPoint` function to change the geometry of an existing point shape.

```
void GXSetPoint(gxShape target, const gxPoint *data);
```

`target` A reference to the point shape whose geometry you want to change.

`data` A pointer to the new point geometry.

DESCRIPTION

The `GXSetPoint` function copies the geometry information from the `data` parameter into the geometry property of the target point shape. If the target shape is not a point shape, this function replaces the target shape with a point shape and sets the shape fill to open-frame fill.

You must provide a pointer to a `gxPoint` structure in the `data` parameter—if you pass `nil` for the `data` parameter, the function posts the error `parameter_is_nil`.

If the target shape is locked (that is, its `gxLockedShape` shape attribute is set), this function posts the error `shape_access_not_allowed`.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

SEE ALSO

For general information about point geometries, see “Point Shapes” on page 2-16.

For the definition of the `gxPoint` structure, see page 2-104.

To create a new point shape, use the `GXNewPoint` function, which is described on page 2-111.

To examine the geometry of an existing point shape, use the `GXGetPoint` function, which is described on page 2-121.

To draw a point geometry without creating a point shape, use the `GXDrawPoint` function, which is described on page 2-158. To draw a point shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXGetLine

You can use the `GXGetLine` function to determine the geometry of an existing line shape.

```
gxLine *GXGetLine(gxShape source, gxLine *data);
```

source A reference to the line shape whose geometry you want to determine.

data A pointer to a `gxLine` structure. The function copies the source shape’s geometry into this structure.

function result A pointer to a copy of the source shape’s geometry.

DESCRIPTION

The `GXGetLine` function copies the geometry information from the source line shape into the `gxLine` structure pointed to by the `data` parameter. As a convenience, this function also returns a pointer to the line geometry as the function result.

If the source shape is not a line shape, this function posts the error code `illegal_type_for_shape`.

You must pass a pointer to a `gxLine` structure in the `data` parameter—if you pass `nil` for this parameter, the function posts the error code `parameter_is_nil`.

Geometric Shapes

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>illegal_type_for_shape</code>	(debugging version)
<code>parameter_is_nil</code>	(debugging version)

SEE ALSO

For general information about line geometries, see “Line Shapes” on page 2-17.

For the definition of the `gxLine` structure, see page 2-105.

To create a new line shape, use the `GXNewLine` function, which is described on page 2-112.

To change the geometry of an existing line shape, use the `GXSetLine` function, which is described in the next section.

To draw a line geometry without creating a line shape, use the `GXDrawLine` function, which is described on page 2-158. To draw a line shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXSetLine

You can use the `GXSetLine` function to change the geometry of a line shape.

```
void GXSetLine(gxShape target, const gxLine *data);
```

`target` A reference to the line shape whose geometry you want to change.

`data` A pointer to the new line geometry.

DESCRIPTION

The `GXSetLine` function copies the geometry information from the `data` parameter into the geometry property of the target line shape. If the target shape is not a line shape, this function replaces the target shape with a line shape and sets the shape fill to open-frame fill.

You must provide a pointer to a `gxLine` structure in the `data` parameter—if you pass `nil` for this parameter, the function posts the error code `parameter_is_nil`.

If the target shape is locked (that is, its `gxLockedShape` shape attribute is set), this function posts the error `shape_access_not_allowed`.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

SEE ALSO

For general information about line geometries, see “Line Shapes” on page 2-17.

For the definition of the `gxLine` structure, see page 2-105.

To create a new line shape, use the `GXNewLine` function, which is described on page 2-112.

To examine the geometry of an existing line shape, use the `GXGetLine` function, which is described on page 2-123.

To draw a line geometry without creating a line shape, use the `GXDrawLine` function, which is described on page 2-158. To draw a line shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXGetCurve

You can use the `GXGetCurve` function to determine the geometry of an existing curve shape.

```
gxCurve *GXGetCurve(gxShape source, gxCurve *data);
```

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

data A pointer to a `gxCurve` structure. The function copies the source shape’s geometry into this structure.

function result A pointer to a copy of the source shape’s geometry.

DESCRIPTION

The `GXGetCurve` function copies the geometry information from the source curve shape into the `gxCurve` structure pointed to by the `data` parameter. As a convenience, this function also returns a pointer to the curve geometry as the function result.

Geometric Shapes

If the source shape is not a curve shape, this function posts the error code `illegal_type_for_shape`.

You must pass a pointer to a `gxCurve` structure in the data parameter—if you pass `nil` for this parameter, the function posts the error code `parameter_is_nil`.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>illegal_type_for_shape</code>	(debugging version)
<code>parameter_is_nil</code>	(debugging version)

SEE ALSO

For general information about curve geometries, see “Curve Shapes” on page 2-18.

For the definition of the `gxCurve` structure, see page 2-105.

To create a new curve shape, use the `GXNewCurve` function, which is described on page 2-113.

To change the geometry of an existing curve shape, use the `GXSetCurve` function, which is described in the next section.

To draw a curve geometry without creating a curve shape object, use the `GXDrawCurve` function, which is described on page 2-159. To draw a curve shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXSetCurve

You can use the `GXSetCurve` function to change the geometry of a curve shape.

```
void GXSetCurve(gxShape target, const gxCurve *data);
```

<code>target</code>	A reference to the curve shape whose geometry you want to change.
<code>data</code>	A pointer to the new curve geometry.

DESCRIPTION

The `GXSetCurve` function copies the geometry information from the data parameter into the geometry property of the target shape. If the target shape is not a curve shape, this function replaces the target shape with a curve shape and sets the shape fill to open-frame fill.

Geometric Shapes

You must provide a pointer to a `gxCurve` structure in the `data` parameter—if you pass `nil` for this parameter, the function posts the error code `parameter_is_nil`.

If the target shape is locked (that is, its `gxLockedShape` shape attribute is set), this function posts the error `shape_access_not_allowed`.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

SEE ALSO

For general information about curve geometries, see “Curve Shapes” on page 2-18.

For the definition of the `gxCurve` structure, see page 2-105.

To create a new curve shape, use the `GXNewCurve` function, which is described on page 2-113.

To examine the geometry of an existing curve shape, use the `GXGetCurve` function, which is described on page 2-125.

To draw a curve geometry without creating a curve shape, use the `GXDrawCurve` function, which is described on page 2-159. To draw a curve shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXGetRectangle

You can use the `GXGetRectangle` function to determine the geometry of an existing rectangle shape.

```
gxRectangle *GXGetRectangle(gxShape source, gxRectangle *data);
```

source A reference to the rectangle shape whose geometry you want to determine.

data A pointer to a `gxRectangle` structure. The function copies the source shape’s geometry into this structure.

function result A pointer to a copy of the source shape’s geometry.

Geometric Shapes

DESCRIPTION

The `GXGetRectangle` function copies the geometry information from the source rectangle shape into the `gxRectangle` data structure pointed to by the `data` parameter. As a convenience, this function also returns a pointer to the rectangle geometry as the function result.

If the source shape is not a rectangle shape, this function posts the error code `illegal_type_for_shape`.

You must pass a pointer to a `gxRectangle` structure in the `data` parameter—if you pass `nil` for this parameter, the function posts the error code `parameter_is_nil`.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`shape_is_nil`

`illegal_type_for_shape` (debugging version)

`parameter_is_nil` (debugging version)

SEE ALSO

For general information about rectangle geometries, see “Rectangle Shapes” on page 2-20.

For the definition of the `gxRectangle` structure, see page 2-106.

To create a new rectangle shape, use the `GXNewRectangle` function, which is described on page 2-114.

To determine the bounding rectangle of a rectangle shape, use the `GXGetShapeBounds` function, which is described in the chapter, “Geometric Operations,” in this book. (The result of the `GXGetShapeBounds` function is an ordered rectangle. Therefore, the result of this function may differ from the geometry of the shape you pass in, even if that shape is a rectangle.)

To change the geometry of an existing rectangle shape, use the `GXSetRectangle` function, which is described in the next section.

To draw a rectangle geometry without creating a rectangle shape, use the `GXDrawRectangle` function, which is described on page 2-160. To draw a rectangle shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXSetRectangle

You can use the `GXSetRectangle` function to change the geometry of a rectangle shape.

```
void GXSetRectangle(gxShape target, const gxRectangle *data);
```

`target` A reference to the rectangle shape whose geometry you want to change.

`data` A pointer to the new rectangle geometry.

DESCRIPTION

The `GXSetRectangle` function copies the geometry information from the `data` parameter into the geometry property of the target shape. If the target shape is not a rectangle shape, this function replaces the target shape with a rectangle shape and sets the shape fill to closed-frame fill if it was originally open-frame fill.

If the target shape is not a rectangle shape, this function posts the error code `illegal_type_for_shape`.

You must provide a pointer to a `gxRectangle` structure in the `data` parameter—if you pass `nil` for this parameter, the function posts the error code `parameter_is_nil`.

If the target shape is locked (that is, its `gxLockedShape` shape attribute is set), this function posts the error `shape_access_not_allowed`.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

SEE ALSO

For general information about rectangle geometries, see “Rectangle Shapes” on page 2-20.

For the definition of the `gxRectangle` structure, see page 2-106.

To create a new rectangle shape, use the `GXNewRectangle` function, which is described on page 2-114.

To examine the geometry of an existing rectangle shape, use the `GXGetRectangle` function, which is described on page 2-127.

To draw a rectangle geometry without creating a rectangle shape, use the `GXDrawRectangle` function, which is described on page 2-160. To draw a rectangle shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXGetPolygons

You can use the `GXGetPolygons` function to determine the geometry of a polygon shape.

```
long GXGetPolygons(gxShape source, gxPolygons *data);
```

`source` A reference to the polygon shape whose geometry you want to determine.

`data` A pointer to a `gxPolygons` data structure. The function copies the source shape's geometry into this structure.

function result The length in bytes of the source shape's geometry.

DESCRIPTION

The `GXGetPolygons` function copies the geometry information from the source polygon shape into the `gxPolygons` structure pointed to by the `data` parameter. As the function result, this function returns the length in bytes of the polygon geometry.

If the source shape is not a polygon shape, this function posts the error code `illegal_type_for_shape`.

You may pass `nil` for the `data` parameter. In this case, the `GXGetPolygons` function still returns the length of the data as the function result, but it does not return the actual data in the `data` parameter.

Typically, to use this function, you go through the following steps:

1. Determine the length of the polygon data by calling this function, passing `nil` for the `data` parameter.
2. Allocate enough memory to hold the polygon data.
3. Call this function again, passing a pointer to the allocated memory in the `data` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`shape_is_nil`

`illegal_type_for_shape` (debugging version)

SEE ALSO

For general information about polygon geometries, see "Polygon Shapes" on page 2-22.

For the definition of the `gxPolygons` structure, see page 2-106.

Geometric Shapes

To create a new polygons shape, use the `GXNewPolygons` function, which is described on page 2-116.

To change the geometry of an existing polygon shape, use the `GXSetPolygons` function, which is described in the next section.

To draw a polygon geometry without creating a polygon shape, use the `GXDrawPolygons` function, which is described on page 2-161. To draw a polygons shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXSetPolygons

You can use the `GXSetPolygons` function to change the geometry of a polygon shape.

```
void GXSetPolygons(gxShape target, const gxPolygons *data);
```

`target` A reference to the polygon shape whose geometry you want to change.

`data` A pointer to the new polygon geometry.

DESCRIPTION

The `GXSetPolygons` function copies the geometry information from the `data` parameter into the geometry property of the target polygon shape. If the target shape is not a polygon shape, this function replaces the target shape with a polygon shape.

If you pass `nil` for the `data` parameter, the function sets the polygon shape to have zero contours.

If the target shape is locked (that is, its `gxLockedShape` shape attribute is set), this function posts the error `shape_access_not_allowed`.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`shape_is_nil`

`number_of_points_exceeds_implementation_limit`

`number_of_contours_exceeds_implementation_limit`

`size_of_polygon_exceeds_implementation_limit`

`count_is_less_than_one`

(debugging version)

`shape_access_not_allowed`

(debugging version)

Geometric Shapes

SEE ALSO

For general information about polygon geometries, see “Polygon Shapes” on page 2-22.

For the definition of the `gxPolygons` structure, see page 2-106.

To create a new polygon shape, use the `GXNewPolygons` function, which is described on page 2-116.

To examine the geometry of an existing polygon shape, use the `GXGetPolygons` function, which is described on page 2-130.

To draw a polygon geometry without creating a polygon shape, use the `GXDrawPolygons` function, which is described on page 2-161. To draw a polygon shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXGetPaths

You can use the `GXGetPaths` function to determine the geometry of a path shape.

```
long GXGetPaths(gxShape source, gxPaths *data);
```

source A reference to the path shape whose geometry you want to determine.

data A pointer to a `gxPaths` structure. The function copies the source shape’s geometry into this structure.

function result The length in bytes of the source shape’s geometry.

DESCRIPTION

The `GXGetPaths` function copies the geometry information from the source path shape into the `gxPaths` structure pointed to by the `data` parameter. As the function result, this function returns the length in bytes of the path geometry.

If the source shape is not a path shape, this function posts the error code `illegal_type_for_shape`.

You may pass `nil` for the `data` parameter. In this case, the `GXGetPaths` function still returns the length of the data, but it does not return the actual data in the `data` parameter.

Geometric Shapes

Typically, to use this function, you go through the following steps:

1. Determine the length of the path data by calling this function, passing `nil` for the `data` parameter.
2. Allocate enough memory to hold the path data.
3. Call this function again, passing a pointer to the allocated memory in the `data` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`illegal_type_for_shape` (debugging version)

SEE ALSO

For general information about path geometries, see “Path Shapes” on page 2-25.

For the definition of the `gxPaths` structure, see page 2-107.

To create a new path shape, use the `GXNewPaths` function, which is described on page 2-117.

To change the geometry of an existing path shape, use the `GXSetPaths` function, which is described in the next section.

To draw a path geometry without creating a path shape, use the `GXDrawPaths` function, which is described on page 2-162. To draw a path shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXSetPaths

You can use the `GXSetPaths` function to change the geometry of a path shape.

```
void GXSetPaths(gxShape target, const gxPaths *data);
```

`target` A reference to the path shape whose geometry you want to change.

`data` A pointer to new path geometry.

Geometric Shapes

DESCRIPTION

The `GXSetPaths` function copies the geometry information from the `data` parameter into the `geometry` property of the target path shape. If the target shape is not a path shape, this function posts the error code `illegal_type_for_shape`.

You must provide a pointer to a `gxPaths` structure in the `data` parameter—if you pass `nil` for this parameter, the function posts the error code `parameter_is_nil`.

If the target shape is locked (that is, its `gxLockedShape` shape attribute is set), this function posts the error `shape_access_not_allowed`.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>count_is_less_than_one</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

SEE ALSO

For general information about path geometries, see “Path Shapes” on page 2-25.

For the definition of the `gxPaths` structure, see page 2-107.

To create a new path shape, use the `GXNewPaths` function, which is described on page 2-117.

To examine the geometry of an existing path shape, use the `GXGetPaths` function, which is described on page 2-132.

To draw a path geometry without creating a path shape, use the `GXDrawPaths` function, which is described on page 2-162. To draw a path shape, use the `GXDrawShape` function, which is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Editing Shape Geometries

The functions described in the previous section, “Getting and Setting Shape Geometries,” allow you to examine and replace entire shape geometries. The functions in this section provide more sophisticated abilities—with these functions, you can examine and edit specific parts of geometries.

For example, the `GXCountShapeContours` function allows you to determine the number of contours in a shape’s geometry. For polygon and path shapes, this number is an integral part of the geometry—it is the first value stored in the geometry; for other geometric shapes, this function simply returns 1.

Similarly, the `GXCountShapePoints` function returns the number of geometric points in a specified contour of a shape’s geometry.

The `GXGetShapeIndex` function returns the geometry index of a specific geometric point given a contour number and the index of the geometric point within the contour. (Remember, each geometric point in a geometry has an geometry index—if you consider a geometry as a list of geometric points starting from the first geometric point of the first contour to the last geometric point of the last contour, the geometry index of a particular geometric point is its position in this list.) You use geometry indexes to specify ranges of geometric points in many of the functions in this section.

You can use the `GXGetShapePoints` function to obtain a copy of a particular range of geometric points from a shape’s geometry, and you can use the `GXSetShapePoints` to replace a particular range of geometric points in a shape’s geometry.

You can use the `GXGetPolygonParts` function to extract a range of geometric points from an existing polygon shape and put them into a new polygon geometry. You can use the `GXSetPolygonParts` function to replace any range of geometric points in an existing polygon shape with any new polygon geometry.

Similarly, you can use the `GetPathsParts` function to extract a range of geometric points from an existing path shape and put them into a new path geometry, and you can use the `SetPathsParts` function to replace any range of geometric points in an existing path shape with any new path geometry.

The `GXGetShapeParts` and `GXSetShapeParts` functions allow the broadest editing control. With the `GXGetShapeParts` function, you can extract any range of geometric points from an existing shape and put them into a new shape. With the `GXSetShapeParts` function, you can replace any range of geometric points in an existing shape with the entire geometry of another shape.

Geometric Shapes

You can apply `GXCountShapeContours`, `GXCountShapePoints`, `GXGetShapeIndex`, `GXGetShapePoints`, `GXSetShapePoints`, `GXGetShapeParts`, and `GXSetShapeParts` functions to other shape types as well as geometric shapes. Information about how they work for geometric shapes is presented in this section. You can find more information about these functions in Chapter 5, “Bitmap Shapes,” and Chapter 6, “Picture Shapes,” and in *Inside Macintosh: QuickDraw GX Typography*.

GXCountShapeContours

You can use the `GXCountShapeContours` function to determine the number of contours in a shape.

```
long GXCountShapeContours(gxShape source);
```

source A reference to the shape whose contours you want to count.

function result The number of contours in the source shape.

DESCRIPTION

The `GXCountShapeContours` function returns as its function result the number of contours in the source shape. For polygon and path shapes, this number indicates the total number of polygon contours or path contours contained in the shape. For points, lines, curves, and rectangles, this function returns the value 1. For empty and full shapes, this function posts the `graphics_type_does_not_have_multiple_contours` error.

If you provide a source shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Always returns 1 as the function result
picture	Returns the number of picture items
text	Returns the number of glyphs
glyph	Returns the number of glyphs
layout	Returns the byte length of the text

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`graphic_type_does_not_have_multiple_contours` (debugging version)

SEE ALSO

For a discussion of contours, see “Shape Geometry” on page 2-9, “Polygon Shapes” on page 2-22, and “Path Shapes” on page 2-25.

To learn how this function works for typographic shape types, see *Inside Macintosh: QuickDraw GX Typography*.

To determine the number of points in a specific contour of a shape, use the `GXCountShapePoints` function, which is described in the next section.

GXCountShapePoints

You can use the `GXCountShapePoints` function to determine the number of geometric points in a specific contour of a shape.

```
long GXCountShapePoints(gxShape source, long contour);
```

`source` A reference to the shape containing the contour.

`contour` The index of the contour whose geometric points you want to count.

function result The number of points in the specified contour of the source shape.

DESCRIPTION

The `GXCountShapePoints` function returns as its function result the number of points in the contour specified by the `contour` parameter of the shape specified by the `source` parameter. If you pass 0 for the `contour` parameter, this function returns the total number of geometric points in the shape.

Geometric Shapes

For the geometric shapes with only one contour—points, lines, curves, and rectangles—you must pass a 0 or a 1 in the `contour` parameter. For polygons and paths shapes, the value you provide for the `contour` parameter must be 0 or greater and must be equal to or less than the actual number of contours in the shape. For empty and full shapes, the function posts a `contour_out_of_range` warning.

If you provide a source shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Returns 1 if the <code>contour</code> parameter is 0 or 1; posts the error <code>contour_out_of_range</code> otherwise
picture	Posts the error <code>graphic_type_does_not_contain_points</code>
text	Returns 1 if the <code>contour</code> parameter is 0 or 1; posts the error <code>contour_out_of_range</code> otherwise
glyph	Returns the number of glyphs in the style run indicated by the <code>contour</code> parameter
layout	Returns the byte length of the style run indicated by the <code>contour</code> parameter

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`graphic_type_does_not_contain_points` (debugging version)

Warnings

`contour_out_of_range`

SEE ALSO

For a discussion of geometric points, see the section “About Geometric Shapes” beginning on page 2-5.

To learn how this function works for typographic shape types, see *Inside Macintosh: QuickDraw GX Typography*.

To determine the number of contours in a shape, use the `GXCountShapeContours` function, which is described on page 2-136.

To determine the index of a particular geometric point within a shape, use the `GXGetShapeIndex` function, which is described in the next section.

GXGetShapeIndex

You can use the `GXGetShapeIndex` function to determine the geometry index of a geometric point.

```
long GXGetShapeIndex(gxShape source, long contour, long vector);
```

<code>source</code>	A reference to the shape containing the desired geometric point.
<code>contour</code>	The index of the contour within the shape containing the geometric point.
<code>vector</code>	The index of the geometric point within that contour.

function result The geometry index of the specified geometric point.

DESCRIPTION

The `GXGetShapeIndex` function returns as its function result the geometry index of the geometric point in the `source` shape's geometry that is identified by the `contour` and `vector` parameters. The indexes you provide in the `contour` and `vector` parameters are 1-based—for example, a value of 1 for the `contour` parameter indicates the first contour, and value of 2 indicates the second contour, and so on.

Each geometric point in a geometry has a geometry index—if you consider a geometry as a list of geometric points starting from the first geometric point of the first contour to the last geometric point of the last contour, the geometry index of a particular geometric point is its position in this list. For example, for a shape with two contours, the first with 10 geometric points and the second with 5 geometric points, this function would return 14 if you set the `contour` parameter to 2 and the `vector` parameter to 4.

For the geometric shapes with only one contour—points, lines, curves, and rectangles—you must pass a 1 in the `contour` parameter. For polygon and path shapes, the value you provide for the `contour` parameter must be greater than 0 and must be equal to or less than the actual number of contours in the shape. Otherwise, the function posts a `contour_out_of_range` warning. Similarly, the value you provide for the `vector` parameter must be equal to or less than the actual number of geometric points in the specified contour, or the function posts an `index_out_of_range_in_contour` warning and returns 0 as the function result.

If you provide a source shape that is an empty shape, a full shape, or a shape that is not one of the geometric shape types, this function posts the error `graphic_type_does_not_have_multiple_contours`.

Geometric Shapes

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>graphic_type_does_not_contain_points</code>	(debugging version)
<code>graphic_type_does_not_have_multiple_contours</code>	(debugging version)

Warnings

<code>contour_out_of_range</code>
<code>index_out_of_range_in_contour</code>

SEE ALSO

For a discussion of geometric points, see the section “About Geometric Shapes” beginning on page 2-5.

To determine the number of contours in a shape, use the `GXCountShapeContours` function, which is described on page 2-136.

To determine the number of geometric points in a contour, use the `GXCountShapePoints` function, which is described on page 2-137.

To copy a range of geometric points from a shape’s geometry, use the `GXGetShapePoints` function, which is described in the next section.

GXGetShapePoints

You can use the `GXGetShapePoints` function to obtain a copy of a range of geometric points from a specified shape.

```
long GXGetShapePoints(gxShape source, long index, long count,
                     gxPoint data[]);
```

<code>source</code>	A reference to the shape containing the desired geometric points.
<code>index</code>	The geometry index of the first geometric point to copy.
<code>count</code>	The number of the geometric points to copy. You may provide the <code>gxSelectToEnd</code> constant for this parameter.
<code>data</code>	A pointer to an array of <code>gxPoint</code> structures. On return, this array contains the copied points.

function result The number of geometric points copied.

DESCRIPTION

The `GXGetShapePoints` function returns in the `data` parameter a copy of the geometric points from the source shape's geometry starting from the geometric point with the geometry index indicated in the `index` parameter.

You provide, in the `count` parameter, the number of geometric points you want copied. The function result is the actual number of points copied. Typically, the value you provide for the `count` parameter is the same as the function result returned by this function. There are two exceptions:

- If you provide too large a value for the `count` parameter—that is, the geometry of the source shape does not have enough geometric points to satisfy your request—this function copies as many geometric points as the shape does have (starting from the geometric point with the geometry index indicated by the `index` parameter). In this case, the function posts a `count_out_of_range` warning, and the function result reflects the actual number of geometric points copied.
- Similarly, if you set the `count` parameter to the `gxSelectToEnd` constant, the function copies as many geometric points as the shape has, starting from the geometric point with the geometry index indicated by the `index` parameter. In this case, the function result reflects the actual number of geometric points copied, but no warning is posted.

Notice that this function returns the copied points as a single point array. If the source shape is a polygon or path shape, the information about which contours contained the geometric points is not retained.

If you want use the `gxSelectToEnd` constant for the `count` parameter, you would typically do the following:

1. Determine the length of the point array by calling this function, passing `nil` for the `data` parameter.
2. Allocate enough memory to hold the point array.
3. Call this function again, passing a pointer to the allocated memory in the `data` parameter.

If you provide an empty or full shape for the source shape, this function posts the error `graphic_type_does_not_contain_points`.

If you provide a source shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Always returns 1 as the function result
picture	Posts the error <code>graphic_type_does_not_contain_points</code>
text	Always returns 1
glyph	Returns the number of glyphs
layout	Always returns 1

Geometric Shapes

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>graphic_type_does_not_contain_points</code>	(debugging version)

Warnings

<code>index_out_of_range_in_contour</code>
<code>count_out_of_range</code>

SEE ALSO

For a discussion of geometric points, see the section “Shape Geometry” beginning on page 2-9.

To learn how this function works for typographic shape types, see *Inside Macintosh: QuickDraw GX Typography*.

To determine the geometry index of a particular geometric point within a shape’s geometry, use the `GXGetShapeIndex` function, which is described on page 2-139.

To replace a range of geometric points in a geometry, use the `GXSetShapePoints` function, which is described in the next section.

GXSetShapePoints

You can use the `GXSetShapePoints` procedure to replace geometric points of a shape.

```
void GXSetShapePoints(gxShape target, long index, long count,
                     const gxPoint data[]);
```

<code>target</code>	A reference to the shape containing the geometric points you want to replace.
<code>index</code>	The geometry index of the first geometric point to replace.
<code>count</code>	The number of the geometric points to replace.
<code>data</code>	An array of new geometric points.

DESCRIPTION

The `GXSetShapePoints` function changes the values of the number of geometric points specified in the `count` parameter, starting with the geometric point indicated by the `index` parameter, to the values specified by the `data` parameter.

Notice that this function replaces geometric points on a point-by-point basis; the number of points in the `data` parameter must match the value of the `count` parameter. You may not use the `gxSelectToEnd` constant for the `count` parameter.

Geometric Shapes

If you provide an empty or full shape for the source shape, this function posts the error `graphic_type_does_not_contain_points`.

If you provide a source shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Sets bitmap position
picture	Posts the error <code>graphic_type_does_not_contain_points</code>
text	Sets position of text shape
glyph	Sets glyph positions corresponding to range indicated by the <code>index</code> and <code>count</code> parameters
layout	Sets position of layout shape

ERRORS, WARNINGS, AND NOTICES

Errors	
<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>graphic_type_does_not_contain_points</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
Warnings	
<code>index_out_of_range_in_contour</code>	
<code>count_out_of_range</code>	

SEE ALSO

For examples that use this function, see “Replacing Geometric Points” beginning on page 2-79.

For a discussion of geometric points, see the section “Shape Geometry” beginning on page 2-9.

To learn how this function works for typographic shape types, see *Inside Macintosh: QuickDraw GX Typography*.

To determine the geometry index of a particular geometric point within a shape, use the `GXGetShapeIndex` function, which is described on page 2-139.

To obtain a copy of a range of geometric points in a geometry, use the `GXGetShapePoints` function, which is described on page 2-140.

GXGetPolygonParts

You can use the `GXGetPolygonParts` function to copy a specified range of geometric points from the geometry of a polygon shape and then put these points into a polygon structure.

```
long GXGetPolygonParts(gxShape source, long index, long count,
                      gxPolygons *data);
```

<code>source</code>	A reference to the polygon shape containing the desired geometric points.
<code>index</code>	The geometry index of the first geometric point to copy.
<code>count</code>	The number of the geometric points to copy. You may provide the <code>gxSelectToEnd</code> constant for this parameter.
<code>data</code>	A pointer to a polygon structure to hold the copied geometric information.

function result The number of bytes required to hold the information returned in the `data` parameter.

DESCRIPTION

The `GXGetPolygonParts` function copies geometry information from the source polygon shape into the polygon structure specified by the `data` parameter. This function copies all of the geometry information starting with the geometric point indicated by the `index` parameter and continuing for as many geometric points as indicated by the `count` parameter. This function copies the values of the indicated geometric points and retains the information about contour breaks from the original geometry. The function result is the length in bytes of the information returned in the `data` parameter.

Both the `index` and the `count` parameters must be greater than 0, although you can provide the `gxSelectToEnd` constant for the `count` parameter, which indicates that you want a copy of all the geometric points starting with the point indicated by the `index` parameter.

You may pass `nil` for the `data` parameter. In this case, the function still returns the byte length as the function result, but does not copy any geometry information.

Typically, to use this function, you go through these steps:

1. Determine the byte length needed to store the copied geometry information by calling this function, passing `nil` for the `data` parameter.
2. Allocate enough memory to hold the copied geometric information.
3. Call this function again, passing a pointer to the allocated memory in the `data` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

```

out_of_memory
shape_is_nil
illegal_type_for_shape      (debugging version)
index_is_less_than_one      (debugging version)
count_is_less_than_one      (debugging version)

```

Warnings

```

index_out_of_range
count_out_of_range

```

SEE ALSO

For an example that uses this function, see “Editing Polygon Parts” beginning on page 2-82.

For a discussion of polygons, see “Polygon Shapes” on page 2-22.

For the definition of the `gxPolygons` structure, see page 2-106.

For information about other functions that allow you to extract information from shape geometries, see the description of the `GXGetShapePoints` function on page 2-140 and the description of the `GXGetShapeParts` function on page 2-152.

To replace parts of a polygon shape’s geometry, use the `GXSetPolygonParts` function, which is described in the next section.

GXSetPolygonParts

You can use the `GXSetPolygonParts` function to replace a range of geometry information in the geometry of a polygon shape with information from a specified polygon structure.

```

void GXSetPolygonParts(gxShape target, long index, long count,
                      const gxPolygons *data,
                      gxEditShapeFlag flags);

```

<code>target</code>	A reference to the polygon shape whose geometry you want to edit.
<code>index</code>	The geometry index of the first geometric point to replace. A value of 0 indicates that the new information should be inserted after the final geometric point in the target shape’s geometry.
<code>count</code>	The number of the geometric points to replace. A value of 0 indicates that no geometric points should be replaced; instead, the new information is inserted before the geometric point indicated by the <code>index</code> parameter. If you pass the <code>gxSelectToEnd</code> constant for this parameter, all geometric points starting with the geometric point indicated by the <code>index</code> parameter are replaced.

Geometric Shapes

<code>data</code>	A pointer to a polygon structure containing the new geometry information.
<code>flags</code>	A set of flags that determine how the new information is inserted in the existing geometry.

DESCRIPTION

The `GXSetPolygonParts` function replaces geometry information in the target shape's geometry with the information pointed to by the `data` parameter. The `index` and `count` parameters determine what part of the original geometry is replaced. The `flags` parameter determines how the new information is inserted in the geometry.

The `data` parameter contains a pointer to the geometry information to be copied into the target shape's geometry. If you pass the `gxSetToNil` constant for this parameter, no new information is copied in; in this case, this function removes the indicated geometric points instead of replacing them.

The `index` parameter indicates the first geometric point to be replaced. If you pass a value of 0 for this parameter, no geometric points are replaced. Instead, this function inserts the new geometry information after the last geometric point of the target shape's original geometry. If you pass 0 for this parameter, you must pass 0 or the `gxSelectToEnd` constant for the `count` parameter.

The `count` parameter indicates how many geometric points in the original geometry should be replaced. If you pass a value of 0 for this parameter, no geometric points are replaced; instead, this function inserts the new geometry information before the geometric point indicated by the `index` parameter. If you pass the `gxSelectToEnd` constant for this parameter, the function replaces all geometric points in the original geometry starting with the geometric point indicated by the `index` parameter.

When this function inserts the new geometry information, it retains the contour breaks contained in the `gxPolygons` structure specified by the `data` parameter. For example, if you provide a `gxPolygons` structure that contains two contours, the break between those contours remains when the new geometric points are inserted in the target shape's geometry.

The `flags` parameter indicates how you want the function to merge the first geometric point and the last geometric point of the `gxPolygons` structure into the target shape's geometry. The possible flags are

<code>gxBreakNeitherEdit</code>	= 0
<code>gxBreakLeftEdit</code>	= 0x01
<code>gxBreakRightEdit</code>	= 0x02
<code>gxRemoveDuplicatePoints</code>	= 0x04

Geometric Shapes

The `gxBreakNeitherEdit` value indicates that the first geometric point of the `gxPolygons` structure should be merged into the preceding contour of the target shape's geometry and the final geometric point of the `gxPolygons` structure should be merged into the following contour.

The `gxBreakLeftEdit` flag indicates that the first geometric point of the `gxPolygons` structure should begin a new contour in the target shape's geometry. The `gxBreakRightEdit` flag indicates that the geometric point in the target shape that follows the final geometric point of the `gxPolygons` structure (after the new information is inserted) should begin a new contour.

The `gxRemoveDuplicatePoints` flag indicates that this function should, when inserting the information from the `gxPolygons` structure, remove the first geometric point of this structure if it exactly matches the preceding geometric point. Similarly, this flag indicates that the final geometric point of the `gxPolygons` structure should be removed if it exactly matches the subsequent geometric point in the target shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>illegal_type_for_shape</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>index_is_less_than_zero</code>	(debugging version)
<code>count_is_less_than_zero</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

Warnings

<code>index_out_of_range</code>
<code>count_out_of_range</code>

SEE ALSO

For an example that uses this function, see “Editing Polygon Parts” beginning on page 2-82.

For a discussion of polygons, see “Polygon Shapes” on page 2-22.

For the definition of the `gxPolygons` structure, see page 2-106.

For information about other functions that allow you to edit information in shape geometries, see the description of the `GXSetShapePoints` function on page 2-142 and the description of the `GXSetShapeParts` function on page 2-154.

To copy parts of a polygon shape's geometry, use the `GXGetPolygonParts` function, which is described on page 2-144.

GXGetPathParts

You can use the `GXGetPathParts` function to extract a copy of a specified range of geometric points from the geometry of a path shape and put these points into a `gxPaths` structure.

```
long GXGetPathParts(gxShape source, long index, long count,
                    gxPaths *data);
```

<code>source</code>	A reference to the path shape containing the desired geometric points.
<code>index</code>	The geometry index of the first geometric point to copy.
<code>count</code>	The number of geometric points to copy. You may provide the <code>gxSelectToEnd</code> constant for this parameter.
<code>data</code>	A pointer to a <code>gxPaths</code> structure. On return, this structure contains the copied geometric information.

function result The number of bytes required to hold the information returned in the `data` parameter.

DESCRIPTION

The `GXGetPathParts` function copies geometry information from the source path shape into the `gxPaths` structure specified by the `data` parameter. This function copies all of the geometry information starting with the geometric point indicated by the `index` parameter and continuing for as many geometric points as indicated by the `count` parameter. This function copies the values of the indicated geometric points and retains the information about contour breaks from the original geometry, as well as the information about which points are on curve and which are off curve. The function result is the length in bytes of the information returned in the `data` parameter.

Both the `index` and the `count` parameters must be greater than 0, although you can provide the `gxSelectToEnd` constant for the `count` parameter, which indicates that you want a copy of all the geometric points starting with the geometric point indicated by the `index` parameter.

You may pass `nil` for the `data` parameter. In this case, the function still returns the byte length as the function result, but does not copy any geometry information.

Typically, to use this function, you go through these steps:

1. Determine the byte length needed to store the copied geometry information by calling this function, passing `nil` for the `data` parameter.
2. Allocate enough memory to hold the copied geometry information.
3. Call this function again, passing a pointer to the allocated memory in the `data` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`
`illegal_type_for_shape` (debugging version)
`index_is_less_than_one` (debugging version)
`count_is_less_than_one` (debugging version)

Warnings

`index_out_of_range`
`count_out_of_range`

SEE ALSO

For a discussion of paths, see “Path Shapes” on page 2-25.

For the definition of the `gxPaths` structure, see page 2-107.

For information about other functions that allow you to extract information from shape geometries, see the description of the `GXGetShapePoints` function on page 2-140 and the description of the `GXGetShapeParts` function on page 2-152.

To replace parts of a path shape’s geometry, use the `GXSetPathParts` function, which is described in the next section.

GXSetPathParts

You can use the `GXSetPathParts` function to replace a range of geometric points in the geometry of a path shape with the information from a specified `gxPaths` structure.

```
void GXSetPathParts(gxShape target, long index, long count,
                   const gxPaths *data, gxEditShapeFlag flags);
```

<code>target</code>	A reference to the path shape whose geometry you want to edit.
<code>index</code>	The index number of the first geometric point to replace. A value of 0 indicates that the new information should be inserted after the final geometric point in the target shape’s geometry.
<code>count</code>	The number of the geometric points to replace. A value of 0 indicates that no geometric points should be replaced; instead, the new information is inserted before the geometric point specified by the <code>index</code> parameter. If you pass the <code>gxSelectToEnd</code> constant for this parameter, all geometric points from the one specified by the <code>index</code> parameter to the final geometric point are replaced.
<code>data</code>	A pointer to the <code>gxPaths</code> structure containing the new geometry information.
<code>flags</code>	A set of flags that determine how the new information is inserted in the existing geometry.

Geometric Shapes

DESCRIPTION

The `GXSetPathParts` function replaces geometry information in the target shape's geometry with the information pointed to by the `data` parameter. The `index` and `count` parameters determine what part of the original geometry is replaced. The `flags` parameter determines how the new information is inserted in the geometry.

The `data` parameter contains a pointer to the geometry information to be copied into the target shape's geometry. If you pass the `gxSetToNil` constant for this parameter, no new information is copied in; in this case, the `GXSetPathParts` function removes the indicated geometric points instead of replacing them.

The `index` parameter indicates the first geometric point to be replaced. If you pass a value of 0 for this parameter, no geometric points are replaced. Instead, this function inserts the new geometric information after the last geometric point of the target shape's original geometry. If you pass 0 for this parameter, you must pass 0 or the `gxSelectToEnd` constant for the `count` parameter.

The `count` parameter indicates how many geometric points in the original geometry should be replaced. If you pass a value of 0 for this parameter, no geometric points are replaced; instead, this function inserts the new geometry information before the geometric point indicated by the `index` parameter. If you pass the `gxSelectToEnd` constant for this parameter, the function replaces all geometric points in the original geometry starting with the one indicated by the `index` parameter.

When this function inserts the new geometric information, it retains the contour breaks contained in the `gxPaths` structure specified in the `data` parameter. For example, if you provide a `gxPaths` structure that contains two contours, the break between those contours remains when the geometric points are inserted into the target shape's geometry.

The `flags` parameter indicates how you want the function to merge the first geometric point and the last geometric point of the `gxPaths` structure into the target shape's geometry. The possible flags are

```

gxBreakNeitherEdit      = 0
gxBreakLeftEdit         = 0x01
gxBreakRightEdit        = 0x02
gxRemoveDuplicatePoints = 0x04

```

The `gxBreakNeitherEdit` value indicates that the first geometric point of the `gxPaths` structure should be merged into the preceding contour of the target shape's geometry and the final geometric point of the `gxPaths` structure should be merged into the subsequent contour.

Geometric Shapes

The `gxBreakLeftEdit` flag indicates that the first geometric point of the `gxPaths` structure should begin a new contour once inserted in the target shape's geometry. The `gxBreakRightEdit` flag indicates that the geometric point in the target shape that follows the final geometric point of the `gxPaths` structure (after the new information is inserted) should begin a new contour.

The `gxRemoveDuplicatePoints` flag indicates that this function should, when inserting the information from the `gxPaths` structure, remove the first geometric point of this inserted structure if it exactly matches the preceding point in the existing geometry. Similarly, this flag indicates that the final geometric point of the `gxPaths` structure should be removed if it exactly matches the subsequent geometric point in the target shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>illegal_type_for_shape</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>index_is_less_than_zero</code>	(debugging version)
<code>count_is_less_than_zero</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

Warnings

<code>index_out_of_range</code>
<code>count_out_of_range</code>

SEE ALSO

For an example that uses this function, see “Editing Paths Parts” beginning on page 2-91.

For a discussion of paths, see “Path Shapes” on page 2-25.

For the definition of the `gxPaths` structure, see “Path Structures” on page 2-107.

For information about other functions that allow you to edit information in shape geometries, see the description of the `GXSetShapePoints` function on page 2-142 and the description of the `GXSetShapeParts` function on page 2-154.

To copy parts of a path shape's geometry, use the `GXGetPathParts` function, which is described on page 2-148.

GXGetShapeParts

You can use the `GXGetShapeParts` function to extract a copy of a specified range of geometric points from the geometry of one shape and encapsulate it in another shape.

```
gxShape GXGetShapeParts(gxShape source, long index, long count,
                        gxShape destination);
```

<code>source</code>	A reference to the shape containing the desired geometric points.
<code>index</code>	The geometry index of the first geometric point to copy.
<code>count</code>	The number of geometric points to copy. You may provide the <code>gxSelectToEnd</code> constant for this parameter.
<code>destination</code>	A reference to the shape to encapsulate the copied geometry information.

function result A copy of the reference returned in the `destination` parameter.

DESCRIPTION

The `GXGetShapeParts` function copies geometry information from the source shape into the destination shape. This function copies all of the geometry information starting with the geometric point indicated by the `index` parameter and continuing for as many geometric points as indicated by the `count` parameter. This function copies the values of the indicated geometric points and retains the information about contour breaks from the original geometry, as well as the information about which points are on curve and which are off curve. As a convenience, the function returns as its function result a reference to the destination shape.

Both the `index` and the `count` parameters must be greater than 0, although you can provide the `gxSelectToEnd` constant for the `count` parameter, which indicates that you want a copy of all the geometric points (starting with the geometric point indicated by the `index` parameter) in the source shape's geometry.

You may pass `nil` for the `destination` parameter. In this case, the function creates a new shape of the appropriate type and encapsulates the extracted geometry information in this new shape.

Geometric Shapes

If the source shape is one of the geometric shape types, this function returns a geometric shape type, as described in the following table:

Shape type	Action taken
empty	Returns an empty shape
full	Returns a full shape
point	Returns a point shape
line	Returns a point or a line shape, depending on the number of geometric points copied
curve	Returns a point, line, or curve shape, depending on the number of geometric points copied
rectangle	Returns a point or a rectangle shape, depending on the number of geometric points copied
polygon	Always returns a polygon shape, even if only one or two geometric points are copied
path	Always returns a path shape, even if only one, two, or three geometric points are copied

If you provide a source shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts <code>shape_operator_may_not_be_a_bitmap</code> error
picture	Returns the number of picture items
text	Returns the number of glyphs
glyph	Returns the number of glyphs
layout	Returns the byte length of the text

SPECIAL CONSIDERATIONS

If you pass `nil` for the `destination` parameter and no error results, the `GXGetShapeParts` 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 creating and disposing of objects.

Geometric Shapes

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)

Warnings

<code>shape_operator_may_not_be_a_bitmap</code>
<code>index_out_of_range</code>
<code>count_out_of_range</code>

SEE ALSO

For information about other functions that allow you to extract information from shape geometries, see the description of the `GXGetShapePoints` function on page 2-140, the description of the `GXGetPolygonParts` function on page 2-144, and the description of the `GXGetPathParts` function on page 2-148.

To replace parts of a shape's geometry, use the `GXSetShapeParts` function, which is described in the next section.

GXSetShapeParts

You can use the `GXSetShapeParts` function to replace a range of geometric points in a shape's geometry with the information in another shape's geometry.

```
void GXSetShapeParts(gxShape target, long index, long count,
                    gxShape insert, gxEditShapeFlag flags);
```

<code>target</code>	A reference to the shape whose geometry you want to edit.
<code>index</code>	The geometry index of the first geometric point to replace. A value of 0 indicates that the new information should be inserted after the final geometric point in the target shape's geometry.
<code>count</code>	How many geometric points to replace. A value of 0 indicates that no geometric points should be replaced; instead, the new information is inserted before the geometric point specified by the <code>index</code> parameter. If you pass the <code>gxSelectToEnd</code> constant for this parameter, all geometric points from the one specified by the <code>index</code> parameter to the final one are replaced.
<code>insert</code>	A reference to the shape whose geometry you want to insert. You may specify the <code>gxSetToNil</code> constant for this parameter to indicate that you want to delete points from the target shape's geometry.
<code>flags</code>	A set of flags that determine how the new geometry information is inserted in the target shape's geometry.

DESCRIPTION

The `GXSetShapeParts` function replaces geometry information in the target shape's geometry with the geometry information in the shape specified by the `insert` parameter. The `index` and `count` parameters determine what part of the original geometry is replaced. The `flags` parameter determines how the new information is inserted in the geometry.

This function converts the shape type of the target shape to be suitable to hold the information from the inserted shape. For example, if the target shape is a line and the inserted shape is a rectangle, this function converts the target shape to a polygon shape before inserting the rectangle.

If the target shape is a rectangle, you may only insert information before both geometric points, after both geometric points, or in place of both geometric points.

You may add any shape to an empty target shape—the result will be identical to the inserted shape. You may also add any shape to a full target shape, but the result will also be a full shape.

The `index` parameter indicates the first geometric point to be replaced. If you pass a value of 0 for this parameter, no geometric points are replaced. Instead, this function inserts the new geometry information after the last geometric point of the target shape's original geometry. If you pass a 0 for this parameter, you must pass a 0 or the `gxSelectToEnd` constant for the `count` parameter.

The `count` parameter indicates how many geometric points in the original geometry should be replaced. If you pass a value of 0 for this parameter, no geometric points are replaced; instead, this function inserts the new geometry information before the geometric point indicated by the `index` parameter. If you pass the `gxSelectToEnd` constant for this parameter, the function replaces all geometric points in the original geometry starting with the geometric point indicated by the `index` parameter.

When this function inserts the new geometry information, it retains the contour breaks contained in the inserted shape's geometry. For example, if you provide a path shape for the inserted shape that contains two contours, the break between those contours remains when the geometric points are inserted into the target shape's geometry.

The `flags` parameter indicates how you want the function to merge the first geometric point and the last geometric point of the inserted shape's geometry into the target shape's geometry. The possible flags are:

<code>gxBreakNeitherEdit</code>	= 0
<code>gxBreakLeftEdit</code>	= 0x01
<code>gxBreakRightEdit</code>	= 0x02
<code>gxRemoveDuplicatePoints</code>	= 0x04

The `gxBreakNeitherEdit` value indicates that the first geometric point of the inserted shape's geometry should be merged into the preceding contour of the target shape's geometry and the final geometric point of the inserted shape's geometry should be merged into the subsequent contour.

Geometric Shapes

The `gxBreakLeftEdit` flag indicates that the first geometric point of the inserted shape's geometry should begin a new contour once inserted in the target shape's geometry. The `gxBreakRightEdit` flag indicates that the geometric point in the target shape that follows the final geometric point of the inserted shape's geometry (after the new information is inserted) should begin a new contour.

The `gxRemoveDuplicatePoints` flag indicates that this function should, when inserting the information from the inserted shape's geometry, remove the first geometric point of this inserted geometry if it exactly matches the preceding point in the existing geometry. Similarly, this flag indicates that the final geometric point of the inserted shape's geometry should be removed if it exactly matches the subsequent geometric point in the target shape's geometry.

If you provide a source shape that is a full shape, this function returns a full shape in the `destination` parameter.

If you provide a source shape that is not one of the geometric shape types, this function performs the actions described in the following table:

Shape type	Action taken
bitmap	Posts the error <code>shape_operator_may_not_be_a_bitmap</code>
picture	Calls the <code>GXSetPictureParts</code> function
text	Calls the <code>GXSetTextParts</code> function
glyph	Calls the <code>GXSetGlyphParts</code> function
layout	Calls the <code>GXSetLayoutParts</code> function

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>functionality_unimplemented</code>	(debugging version)
<code>rectangles_cannot_be_inserted_into</code>	(debugging version)
<code>shape_operator_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)

Warnings

<code>index_out_of_range</code>
<code>count_out_of_range</code>
<code>picture_cannot_contain_itself</code>

Notices

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

SEE ALSO

For an example of this function, see “Editing Shape Parts” beginning on page 2-93.

To learn how this function works for typographic shape types, see *Inside Macintosh: QuickDraw GX Typography*.

For information about other functions that allow you to edit information in shape geometries, see the description of the `GXSetShapePoints` function on page 2-142, the description of the `GXSetPolygonParts` function on page 2-145, and the description of the `GXSetPathParts` function on page 2-149.

To copy parts of a shape’s geometry, use the `GXGetShapeParts` function, which is described on page 2-152.

Drawing Geometric Shapes

The QuickDraw GX drawing functions compile all of the information in a shape’s properties, and the properties of its style, ink, and transform objects, and produce a graphic image. Therefore, to understand how these functions draw geometric shapes, you need to be familiar with much of the information in *Inside Macintosh: QuickDraw GX Objects*, as well as much of the information in this chapter and in the next chapter, “Geometric Styles.” The function descriptions in this section give an overview of the process these functions use to draw geometric shapes.

If you want to draw a geometric shape without creating a shape object—that is, just given a geometry—you can use the `GXDrawPoint`, `GXDrawLine`, `GXDrawCurve`, `GXDrawRectangle`, `GXDrawPolygons`, or `GXDrawPaths` functions, which are described in this section. These functions create a shape object, initialize it, draw it, and dispose of it; therefore, they do not take advantage of the QuickDraw GX caching mechanism. You should make limited use of these functions—for example, you could use one of these functions if you wanted to draw a particular shape drawn only once.

To draw a shape once you have created a shape object and modified its properties to suit your needs, you can use the `GXDrawShape` function. This function draws all shape types, and is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

When debugging your application, you can use the `GXGetDrawError` function, which is described in the chapter “QuickDraw GX Debugging” in *Inside Macintosh: QuickDraw GX Environment and Utilities*, for hints when a shape fails to draw as expected.

GXDrawPoint

You can use the `GXDrawPoint` function to draw a point without creating a point shape.

```
void GXDrawPoint(const gxPoint *data);
```

`data` A pointer to the point geometry you want to draw.

DESCRIPTION

The `GXDrawPoint` function draws the point geometry specified by the `data` parameter, using the shape fill, style, ink, and transform of the default point shape.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`parameter_is_nil` (debugging version)

SEE ALSO

For examples using this function, see “Creating and Drawing Points” beginning on page 2-29.

For more information about points and the default point shape, see “Point Shapes” on page 2-16.

For the definition of the `gxPoint` structure, see page 2-104.

For more information about drawing shapes, see the description of the `GXDrawShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXDrawLine

You can use the `GXDrawLine` function to draw a line without creating a line shape.

```
void GXDrawLine(const gxLine *data);
```

`data` A pointer to the line geometry you want to draw.

DESCRIPTION

The `GXDrawLine` function draws the line geometry specified by the `data` parameter, using the shape fill, style, ink, and transform of the default line shape.

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
parameter_is_nil (debugging version)

SEE ALSO

For examples using this function, see “Creating and Drawing Lines” beginning on page 2-36.

For more information about lines and the default line shape, see “Line Shapes” on page 2-17.

For the definition of the `gxLine` structure, see page 2-105.

For more information about drawing shapes, see the description of the `GXDrawShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXDrawCurve

You can use the `GXDrawCurve` function to draw a curve without creating a curve shape.

```
void GXDrawCurve(const gxCurve *data);
```

`data` A pointer to the curve geometry you want to draw.

DESCRIPTION

The `GXDrawCurve` function draws the curve geometry specified by the `data` parameter, using the shape fill, style, ink, and transform of the default curve shape.

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
parameter_is_nil (debugging version)

SEE ALSO

For examples using this function, see “Creating and Drawing Curves” beginning on page 2-41.

For more information about curves and the default curve shape, see “Curve Shapes” on page 2-18.

For the definition of the `gxCurve` structure, see page 2-105.

For more information about drawing shapes, see the description of the `GXDrawShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXDrawRectangle

You can use the `GXDrawRectangle` function to draw a rectangle without creating a rectangle shape.

```
void GXDrawRectangle(const gxRectangle *data, gxShapeFill fill);
```

`data` A pointer to the rectangle geometry you want to draw.

`fill` The shape fill to use when drawing the rectangle.

DESCRIPTION

The `GXDrawRectangle` function draws the rectangle geometry specified by the `data` parameter, using the shape fill specified by the `fill` parameter, and the style, ink, and transform of the default rectangle shape.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`parameter_is_nil`

`shapeFill_is_not_allowed` (debugging version)

(debugging version)

SEE ALSO

For examples using this function, see “Creating and Drawing Rectangles” beginning on page 2-43.

For more information about rectangles and the default rectangle shape, see “Rectangle Shapes” on page 2-20.

For the definition of the `gxRectangle` structure, see page 2-106.

For more information about drawing shapes, see the description of the `GXDrawShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXDrawPolygons

You can use the `GXDrawPolygons` function to draw polygon contours without creating a polygon shape.

```
void GXDrawPolygons(const gxPolygons *data, gxShapeFill fill);
```

`data` A pointer to the polygon geometry you want to draw.

`fill` The shape fill to use when drawing the polygon contours.

DESCRIPTION

The `GXDrawPolygons` function draws the polygon geometry specified by the `data` parameter, using the shape fill specified by the `fill` parameter, and the style, ink, and transform of the default polygon shape.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`parameter_is_nil` (debugging version)

SEE ALSO

For more information about polygons and the default polygon shape, see “Polygon Shapes” on page 2-22.

For the definition of the `gxPolygons` structure, see page 2-106.

For examples using this function, see “Creating and Drawing Polygons” beginning on page 2-45.

For more information about drawing shapes, see the description of the `GXDrawShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

GXDrawPaths

You can use the `GXDrawPaths` function to draw path contours without creating a path shape.

```
void GXDrawPaths(const gxPaths *data, gxShapeFill fill);
```

`data` A pointer to the path geometry you want to draw.

`fill` The shape fill to use when drawing the path contours.

DESCRIPTION

The `GXDrawPaths` function draws the path geometry specified by the `data` parameter, using the shape fill specified by the `fill` parameter, and the style, ink, and transform of the default path shape.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`parameter_is_nil` (debugging version)

SEE ALSO

For more information about paths and the default path shape, see “Path Shapes” on page 2-25.

For the definition of the `gxPaths` structure, see page 2-107.

For examples using this function, see “Creating and Drawing Paths” beginning on page 2-55.

For more information about drawing shapes, see the description of the `GXDrawShape` function in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

Summary of Geometric Shapes

Constants and Data Types

The Point Structure

```
struct gxPoint {  
    Fixed    x;  
    Fixed    y;  
};
```

The Line Structure

```
struct gxLine {  
    struct gxPoint first;  
    struct gxPoint last;  
};
```

The Curve Structure

```
struct gxCurve {  
    struct gxPoint first;  
    struct gxPoint control;  
    struct gxPoint last;  
};
```

The Rectangle Structure

```
struct gxRectangle {  
    Fixed    left;  
    Fixed    top;  
    Fixed    right;  
    Fixed    bottom;  
};
```

Polygon Structures

```
struct gxPolygon {
    long          vectors;
    struct gxPoint vector[gxAnyNumber];
};

struct gxPolygons {
    long          contours;
    struct gxPolygon contour[gxAnyNumber];
};
```

Path Structures

```
struct gxPath {
    long          vectors;
    long          controlBits[gxAnyNumber];
    struct gxPoint vector[gxAnyNumber];
};

struct gxPaths {
    long          contours;
    struct gxPath contour[gxAnyNumber];
};
```

Functions

Creating Geometric Shapes

```
gxShape GXNewPoint          (const gxPoint *data);
gxShape GXNewLine           (const gxLine *data);
gxShape GXNewCurve          (const gxCurve *data);
gxShape GXNewRectangle      (const gxRectangle *data);
gxShape GXNewPolygons       (const gxPolygons *data);
gxShape GXNewPaths          (const gxPaths *data);
```


Getting and Setting Shape Geometries

```

gxPoint *GXGetPoint      (gxShape source, gxPoint *data);
void GXSetPoint          (gxShape target, const gxPoint *data);
gxLine *GXGetLine        (gxShape source, gxLine *data);
void GXSetLine           (gxShape target, const gxLine *data);
gxCurve *GXGetCurve       (gxShape source, gxCurve *data);
void GXSetCurve           (gxShape target, const gxCurve *data);
gxRectangle *GXGetRectangle (gxShape source, gxRectangle *data);
void GXSetRectangle       (gxShape target, const gxRectangle *data);
long GXGetPolygons        (gxShape source, gxPolygons *data);
void GXSetPolygons        (gxShape target, const gxPolygons *data);
long GXGetPaths           (gxShape source, gxPaths *data);
void GXSetPaths           (gxShape target, const gxPaths *data);

```

Editing Shape Geometries

```

long GXCountShapeContours (gxShape source);
long GXCountShapePoints   (gxShape source, long contour);
long GXGetShapeIndex      (gxShape source, long contour, long vector);
long GXGetShapePoints     (gxShape source, long index, long count,
                           gxPoint data[]);
void GXSetShapePoints     (gxShape target, long index, long count,
                           const gxPoint data[]);
long GXGetPolygonParts    (gxShape source, long index, long count,
                           gxPolygons *data);
void GXSetPolygonParts    (gxShape target, long index, long count,
                           const gxPolygons *data,
                           gxEditShapeFlag flags);
long GXGetPathParts       (gxShape source, long index, long count,
                           gxPaths *data);
void GXSetPathParts       (gxShape target, long index, long count,
                           const gxPaths *data, gxEditShapeFlag flags);
gxShape GXGetShapeParts   (gxShape source, long index, long count,
                           gxShape destination);
void GXSetShapeParts      (gxShape target, long index, long count,
                           gxShape insert, gxEditShapeFlag flags);

```

Geometric Shapes

Drawing Geometric Shapes

```
void GXDrawPoint          (const gxPoint *data);  
void GXDrawLine           (const gxLine *data);  
void GXDrawCurve          (const gxCurve *data);  
void GXDrawRectangle      (const gxRectangle *data, gxShapeFill fill);  
void GXDrawPolygons       (const gxPolygons *data, gxShapeFill fill);  
void GXDrawPaths          (const gxPaths *data, gxShapeFill fill);
```