

## Shape Objects

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

Before reading this chapter, you should be familiar with the information in the chapter “Introduction to QuickDraw GX” in this book. For more information on graphic shapes, see *Inside Macintosh: QuickDraw GX Graphics*. For more information on typographic shapes see *Inside Macintosh: QuickDraw GX Typography*. Additional information relevant to the storage of shape objects is in the stream format chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

This chapter introduces the concept of a QuickDraw GX shape, and describes shape objects and their properties. It then shows how to use general QuickDraw GX shape-manipulation functions to

- create and manipulate shape objects
- manipulate shape object properties
- directly manipulate shape geometry
- draw and hit-test shapes

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

## About QuickDraw GX Shapes

---

Shapes are fundamental to the QuickDraw GX object architecture. To draw or print in QuickDraw GX requires creating and manipulating QuickDraw GX shapes. A **shape** is a drawable graphic or typographic entity that you create with QuickDraw GX objects.

Shapes come in two general categories: graphic and typographic. Graphic shapes are further subdivided into geometric shapes (points, lines, rectangles, polygons, and so on), bitmap shapes, and picture shapes. Typographic shapes are subdivided into text shapes, glyph shapes, and layout shapes. Table 2-1 on page 2-9 describes all the types of shapes recognized by QuickDraw GX.

This chapter discusses only shapes in general. The QuickDraw GX object architecture allows you to perform many operations on a shape without regard for what type of shape it is; those are the operations described here.

In the QuickDraw GX architecture, every shape includes four objects:

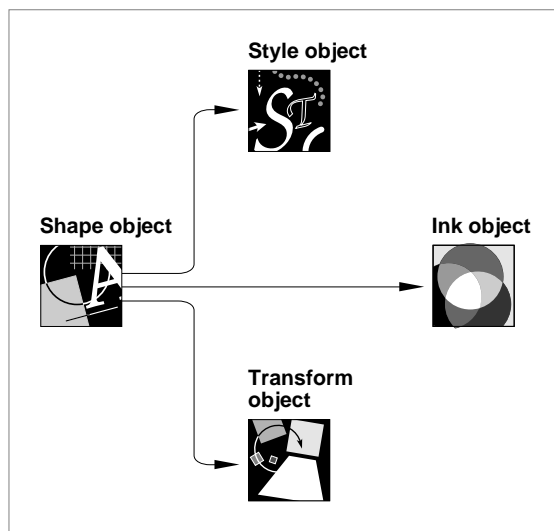
- **Shape object.** A shape object describes the geometric structure or text content of a shape and contains references to the other objects that make up the shape.
- **Style object.** A style object defines much of the appearance of a shape, such as the size of the pen with which it is drawn or the size of its text. See the chapter “Style Objects” in this book for more information.

## Shape Objects

- **Ink object.** An ink object defines the color and transfer mode to use when drawing a shape. See the chapter “Ink Objects” in this book for more information.
- **Transform object.** A transform object defines how the appearance of a shape is altered (such as by clipping, scaling, or rotation) when it is drawn, and how the shape responds to mouse clicks. A transform object also contains references to the view port objects that describe where the shape is drawn. See the chapter “Transform Objects” in this book for more information.

Figure 2-1 shows the four objects used to represent a QuickDraw GX shape.

**Figure 2-1** Basic components of a QuickDraw GX shape



The interface to each of these object types is entirely procedural—you cannot in most cases access any information in the objects directly. You must manipulate the items of information in an object, called the *properties* of the object, using QuickDraw GX functions.

The rest of this chapter describes the data types and functions you can use to create and manipulate shape objects and their properties.

### Terminology Note

A QuickDraw GX *shape* is considered to be the combination of four objects just described. A *shape object* is one of the objects that make up the shape; it defines, among other characteristics, the shape’s *geometry*, which is the description of the specific dimensions and location of the kind of shape (line, curve, rectangle, and so on) that is to be drawn. ♦

## About Shape Objects

This section describes the contents of the shape object and summarizes some of the main tasks you can perform with shapes.

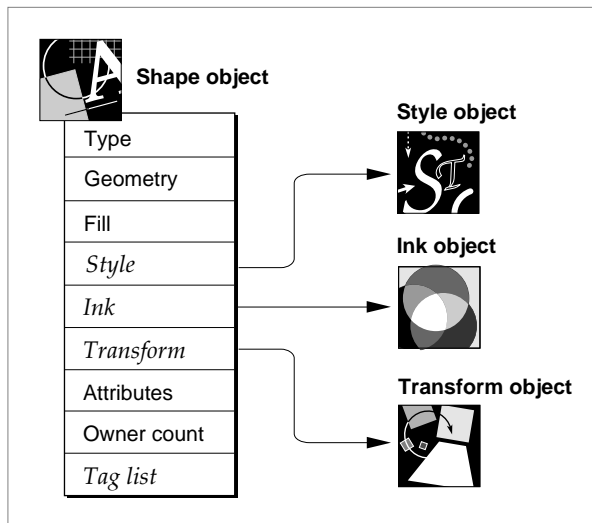
QuickDraw GX identifies an individual shape object through a shape *reference*. To obtain information about a shape object, you must send its reference as a parameter to a QuickDraw GX function (except that you can determine if two references identify the same shape object simply by comparing them for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

Shapes are device-independent. Their location, resolution, color, and other properties are not constrained by the characteristics of the display device to which they are drawn.

### Shape Properties

The properties of a shape object for the most part define the basic geometric characteristics of the shape. Shape objects have nine accessible properties, as shown in Figure 2-2. Note that, because a shape is an object and not a data structure, the order of the properties as shown in Figure 2-2 is completely arbitrary. Properties in italics are references to other objects.

**Figure 2-2** The shape object and its properties



## Shape Objects

The first six properties are specific to shape objects alone. They determine a shape's geometric type, geometric structure, fill, and references to other objects:

- **Type.** A value that specifies what type of geometry the shape object has. The different shape types include point, line, rectangle, text, glyphs, and so on. The section “Shape Type” beginning on page 2-9 describes the different shape types, and “Getting and Setting a Shape Object's Type, Fill, and Attributes” beginning on page 2-28 discusses how to manipulate this property.
- **Geometry.** A set of values that describes the specific graphic structure of the shape. For example, the geometry of a point shape specifies the two coordinates of the point. The geometry of a text shape specifies the sequence of characters or glyphs that it contains. *Inside Macintosh: QuickDraw GX Graphics* discusses the geometries of graphic shapes and *Inside Macintosh: QuickDraw GX Typography* discusses the geometries of typographic shapes. See also Figure 2-3 on page 2-12 of this chapter for a summary of shape geometries. The geometry property differs from other properties in one important respect: you can edit it directly. See “Directly Manipulating a Shape's Geometry” beginning on page 2-34 for more details.
- **Fill.** A value that determines how a shape is filled or framed when drawn. QuickDraw GX provides a number of different ways of filling a shape. For example, a rectangle shape might have a **solid fill**, which indicates that the shape represents a solid rectangle—that is, the entire area enclosed by the sides of the rectangle is included in the shape. Alternatively, a rectangle shape might have a **framed fill**, which indicates that the shape represents a hollow rectangle—only the lines connecting the rectangle's corners are included in this shape. The section “Shape Fill” beginning on page 2-13 discusses types of shape fills, and the section “Getting and Setting a Shape Object's Type, Fill, and Attributes” beginning on page 2-28 discusses how to manipulate the shape fill property of a shape object.
- **Style, ink, and transform object references.** References to the style object, ink object, and transform object that are needed to complete the specification of the shape. The section “Getting and Setting a Shape Object's Style, Ink, and Transform” beginning on page 2-30 discusses how to manipulate these references.

The remaining three shape properties are common to many QuickDraw GX objects (including styles, inks, and others):

- **Attributes.** A group of flags that control certain aspects of the behavior of the object. For a shape object, these flags allow you to specify where QuickDraw GX stores the shape object and how editing operations affect the shape object. For example, the `gxMemoryShape` attribute specifies that QuickDraw GX should avoid writing the shape object out to storage, and the `gxMapTransformShape` attribute indicates that certain editing operations, such as the `GXMoveShape` function, are to affect the data in the shape's transform object rather than the data in the shape itself. The section “Shape Attributes” beginning on page 2-16 describes the shape attribute flags, and the section “Getting and Setting a Shape Object's Type, Fill, and Attributes” beginning on page 2-28 discusses how to manipulate the attributes property of a shape object.

## Shape Objects

- **Owner count.** A number that indicates how many references to the object exist. The chapter “Introduction to QuickDraw GX” in this book includes general information about owner counts, and “Manipulating a Shape Object’s Owner Count” beginning on page 2-31 describes when and how to examine and alter a shape object’s owner count.
- **Tag list.** A list of references to custom information about the object, stored in private data structures called *tag objects*. The chapter “Tag Objects” in this book describes tag objects in general and how you can use them to add custom information to objects. The section “Getting and Setting a Shape Object’s Tag References” beginning on page 2-32 discusses how to manipulate the tag objects associated with a shape object.

## Shape Type

A shape object’s type property specifies what sort of geometry the shape has. Table 2-1 lists the defined constants for each shape type and describes what each one means. (Note that the empty and full shape types are unusual, in that they have no specific geometry; they are used for specialized operations other than drawing.) The constants are defined in the `gxShapeTypes` enumeration.

**Table 2-1** Shape types

Constant	Value	Explanation
<code>gxEmptyType</code>	1	An empty shape. It has no geometry, no contents, and no bounds. The intersection of two shapes that do not touch is the empty shape. You can use empty shapes as a starting point for collecting graphic elements. This shape type is described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
<code>gxPointType</code>	2	A point shape. Its geometry contains two fixed-point coordinate values representing the location of the point. This shape type is described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
<code>gxLineType</code>	3	A line shape. Its geometry contains two point geometries—the starting point and the ending point. This shape type is described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
<code>gxCurveType</code>	4	A curve shape. Its geometry contains three point geometries—a starting point, an ending point, and a control point—which together describe a quadratic Bézier curve. This shape type is described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .

*continued*

## Shape Objects

**Table 2-1** Shape types (continued)

Constant	Value	Explanation
<code>gxRectangleType</code>	5	A rectangle shape. Its geometry contains four fixed-point values—representing the coordinates of the left, top, right, and bottom corners of the rectangle. This shape type is described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
<code>gxPolygonType</code>	6	A polygon shape. Its geometry includes any number of separate multiple-point polygon contours, each contour consisting of straight line segments connecting its points. This shape type is described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
<code>gxPathType</code>	7	A path shape. Its geometry includes any number of separate multiple-point path contours, each contour consisting of straight or curved line segments connecting its points. This shape type is described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
<code>gxBitmapType</code>	8	A bitmap shape. Its geometry contains information about the bitmap's size, shape, and color, as well as the pixel image itself. This shape type is described in the bitmap shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
<code>gxTextType</code>	9	A text shape. Its geometry contains a string of characters to be drawn with uniform stylistic properties such as font family, font size, and style. This shape type is described in the text shapes chapter of <i>Inside Macintosh: QuickDraw GX Typography</i> .
<code>gxGlyphType</code>	10	A glyph shape. Its geometry contains a string of glyphs, each of which may have its own typestyle when drawn. This shape type is described in the glyph shapes chapter of <i>Inside Macintosh: QuickDraw GX Typography</i> .
<code>gxLayoutType</code>	11	A layout shape. Its geometry contains a string of characters plus sophisticated formatting information that affects how the text is displayed. This shape type is described in the layout shapes chapter of <i>Inside Macintosh: QuickDraw GX Typography</i> .

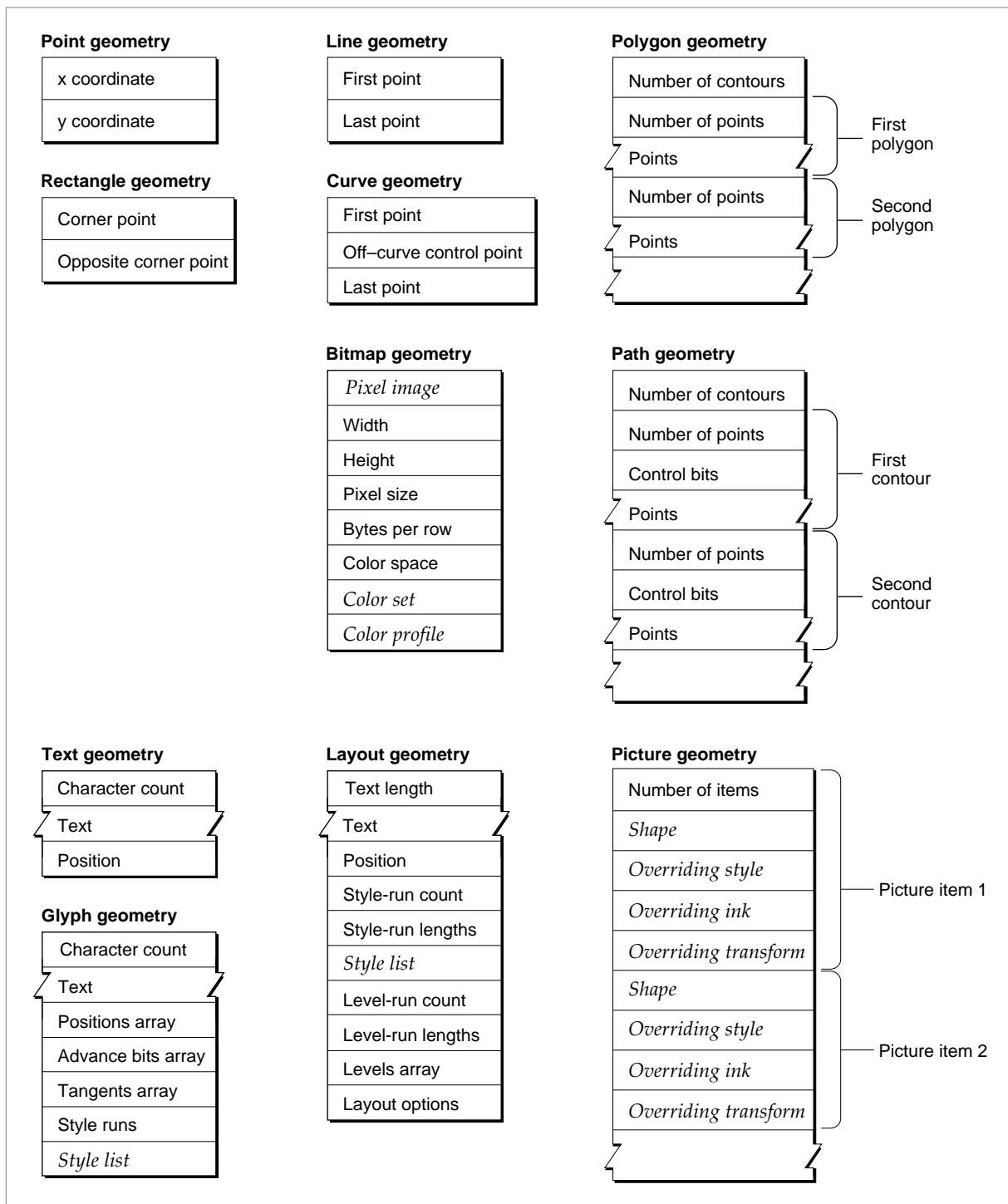
Table 2-1      Shape types (continued)

Constant	Value	Explanation
<code>gxFullType</code>	12	A full shape. It encompasses all of the QuickDraw GX coordinate space. Inverting an empty shape produces a full shape. The full shape contains every other shape and no other shape contains the full shape. You can use full shapes to specify the largest possible clip area for maximum visibility when drawing. This shape type is described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
<code>gxPictureType</code>	13	A picture shape. It is a collection of other shapes, including possibly other picture shapes. This shape type is described in the picture shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .

### Shape Geometry

Most shape geometries are defined in terms of points in a coordinate space. QuickDraw GX coordinates are pairs of fixed-point numbers (of type `Fixed`), as defined in the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. QuickDraw GX coordinate spaces are described in the chapter “View-Related Objects” in this book.

Figure 2-3 summarizes the contents of the geometry property for each type of QuickDraw GX shape (except empty and full types, which have no geometry). In the figure, elements of the geometry that are references (or arrays of references) to other objects are shown in italics. For a diagram showing all the properties of the basic QuickDraw GX objects, see the chapter “Introduction to Objects” in this book. For a diagram showing all the properties of the printing objects, see the introductory chapter of *Inside Macintosh: QuickDraw GX Printing*.

**Figure 2-3** Shape geometry for each type of QuickDraw GX shape



Shape Objects

The structures of individual shape geometries are specific to each shape type and thus are not described here. See the chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography* specified in Table 2-1 of the previous section for more information.

Nevertheless, some of the functions that affect shape geometry are common to all types of shapes, and are therefore described in this chapter. The section “Copying the Geometry From One Shape to Another” beginning on page 2-29 discusses how to copy the geometry between shapes. The section “Directly Manipulating a Shape’s Geometry” beginning on page 2-34 discusses how to get direct access to a shape’s geometry—as a data structure rather than as an object property—in order to modify it. Also, Figure 2-3 on page 2-12 of this chapter summarizes the contents of the geometry of each type of QuickDraw GX shape.

Shape Fill

Each shape object has a fill property. The shape fill specifies how QuickDraw GX interprets the geometry of the shape: how the shape is drawn, how the shape is hit-tested, and how certain geometric operations, like intersection or union, interpret the shape. Table 2-2 lists the defined constants for shape fill and describes what each one means. (Note that some shape fills have two or more equivalent names.) The constants are defined in the `gxShapeFills` enumeration.

Table 2-2      Shape fills

Constant	Value	Explanation
<code>gxNoFill</code>	0	No fill—the shape is not filled at all. QuickDraw GX does not draw a shape with this shape fill and you may not perform geometric operations on it. You can use this shape fill to temporarily hide shapes or to prevent parts of a picture from drawing.
<code>gxOpenFrameFill</code>	1	Open-frame fill—the shape is outlined instead of filled. With this shape fill, QuickDraw GX interprets the shape as a connected series of straight or curved lines from the starting point of the shape geometry to the ending point (but not back to the starting point again).
<code>gxFrameFill</code>	1	Framed fill (same as <code>gxOpenFrameFill</code> ).
<code>gxClosedFrameFill</code>	2	Closed-frame fill—the shape is outlined instead of filled. As with the open-frame fill, QuickDraw GX interprets the shape as a series of lines (or curves) from the starting point of the shape geometry to the ending point. However, in this case, QuickDraw GX also includes a line (or curve) from the ending point to the starting point, thus closing the contour.

*continued*

## Shape Objects

**Table 2-2** Shape fills (continued)

Constant	Value	Explanation
<code>gxHollowFill</code>	2	Hollow fill (same as <code>gxClosedFrameFill</code> ).
<code>gxEvenOddFill</code>	3	Even-odd fill—the shape is filled using the even-odd rule. See Figure 2-4 for an illustration of this rule; see <i>Inside Macintosh: QuickDraw GX Graphics</i> for further explanation.
<code>gxSolidFill</code>	3	Solid fill (same as <code>gxEvenOddFill</code> ).
<code>gxWindingFill</code>	4	Winding fill—the shape is filled using the winding-number rule. See Figure 2-4 on page 2-14 for an illustration of this rule; see <i>Inside Macintosh: QuickDraw GX Graphics</i> for further explanation.
<code>gxInverseEvenOddFill</code>	5	Inverse even-odd fill—the shape is filled in an opposite manner from the even-odd rule; everything <i>not</i> filled using the even-odd rule is filled using this rule. See <i>Inside Macintosh: QuickDraw GX Graphics</i> for further explanation.
<code>gxInverseSolidFill</code>	5	Inverse solid fill (same as <code>gxInverseEvenOddFill</code> ).
<code>gxInverseFill</code>	5	Inverse fill (same as <code>gxInverseEvenOddFill</code> ).
<code>gxInverseWindingFill</code>	6	Inverse winding fill—the shape is filled using the winding-number rule and then inverted. See <i>Inside Macintosh: QuickDraw GX Graphics</i> for further explanation.

**Figure 2-4** Even-odd and winding fills

Shape Objects

Note that framed fill, hollow fill, and solid fill are alternative names for open-frame fill, closed-frame fill, and even-odd fill, respectively, and that both inverse solid fill and inverse fill are alternate names for inverse even-odd fill.

Not all shape fills make sense for all shape types. Table 2-3 shows the acceptable shape fills for each shape type (the alternative names are not listed). Note that empty and full shapes are permitted to have certain fill types even though they have no geometry.

**Table 2-3** Valid shape fills for each shape type

Shape types	Valid shape fills
Empty	gxNoFill gxInverseEvenOddFill gxInverseWindingFill
Full	gxNoFill gxEvenOddFill gxWindingFill gxInverseEvenOddFill gxInverseWindingFill
Point, line, curve	gxNoFill gxOpenFrameFill
Rectangle	gxNoFill gxClosedFrameFill gxEvenOddFill gxWindingFill gxInverseEvenOddFill gxInverseWindingFill
Polygon, path	any shape fill
Text, glyph, layout	gxNoFill gxEvenOddFill gxWindingFill
Bitmap, picture	gxNoFill gxEvenOddFill

For additional examples of how different shape fills can affect the appearance of different types and geometries of shapes, see the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

## Shape Objects

## Shape Attributes

Each shape object includes a property that is a set of attributes, a group of flags that specify certain aspects of the shape's behavior. Table 2-4 lists the defined shape attribute constants and describes what each one means. The constants are defined in the `gxShapeAttributes` enumeration.

**Table 2-4** Shape attributes

Constant	Value	Explanation
<code>gxNoAttributes</code>	0x0000	No shape attributes are set. You can use this attribute to clear or test against the current value of a shape's attributes.
<code>gxDirectShape</code>	0x0001	QuickDraw GX is to load the shape into directly accessible memory. Set this flag for shape objects that you don't want stored in accelerator card memory, or whose geometric structures you want to manipulate directly (see "Directly Manipulating a Shape's Geometry" beginning on page 2-34). The attributes <code>gxDirectShape</code> and <code>gxRemoteShape</code> are exclusive; do not set them both.
<code>gxRemoteShape</code>	0x0002	QuickDraw GX is to load the shape into remote memory (memory used by an accelerator card), if possible. When this flag is set, the shape might draw faster but you might not be able to edit the shape's geometry directly (see "Directly Manipulating a Shape's Geometry" beginning on page 2-34). The attributes <code>gxRemoteShape</code> and <code>gxDirectShape</code> are exclusive; do not set them both.
<code>gxCachedShape</code>	0x0004	QuickDraw GX is to prepare the shape for the fastest possible drawing by caching it compressed in an offscreen bitmap. (See "Caching Shape Objects" beginning on page 2-27; also, compare this with using the <code>GXCachedShape</code> function, described on page 2-62.)

**Table 2-4** Shape attributes (continued)

Constant	Value	Explanation
gxLockedShape	0x0008	QuickDraw GX is to prohibit changes to the shape's geometry or the shape's disposal. You can use this flag in the debugging version of QuickDraw GX to prevent accidental modification of a shape intended to be used as a constant. When this flag is set, you cannot use the geometry-editing functions described in the geometric shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> and the text, glyph, and layout shapes chapters of <i>Inside Macintosh: QuickDraw GX Typography</i> . However, you can still alter the shape's geometric structure by accessing it directly (see "Directly Manipulating a Shape's Geometry" beginning on page 2-34).
gxGroupShape	0x0010	QuickDraw GX is to group all shapes within this shape as a single shape when hit-testing. This attribute applies to picture shapes only; for more information see the picture shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
gxMapTransformShape	0x0020	QuickDraw GX is to apply shape-transforming operations to the shape's transform object rather than to the shape's geometry. This attribute is set by default for bitmap shapes, picture shapes, and layout shapes. See the chapter "Transform Objects" in this book for more information on applying transformations to shapes.
gxUniqueItemsShape	0x0040	QuickDraw GX is to create a complete copy of each shape added to this picture rather than simply creating a reference to the added shape. This attribute applies to picture shapes only; for more information see the picture shapes chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> .
gxIgnorePlatformShape	0x0080	QuickDraw GX is to treat the codes in the geometry of this shape as glyph codes rather than character codes. This attribute applies to typographic shapes only; for more information see the typographic shapes chapter of <i>Inside Macintosh: QuickDraw GX Typography</i> .

*continued*

## Shape Objects

**Table 2-4** Shape attributes (continued)

Constant	Value	Explanation
<code>gxNoMetricsGridShape</code>	0x0100	QuickDraw GX is not to use hints (special display instructions) provided with the font used for this shape. Set this attribute if you intend to manipulate text as a path shape; otherwise, the hinting can affect the spacing between the contours in the path's geometry and can be undesirable if you want to perform other operations such as scaling. This attribute applies to typographic shapes only; for more information see the typographic shapes chapter of <i>Inside Macintosh: QuickDraw GX Typography</i> .
<code>gxDiskShape</code>	0x0200	QuickDraw GX is to write this shape to disk before all shapes that do not have this attribute set when it needs to unload shapes to minimize memory requirements. The attributes <code>gxDiskShape</code> and <code>gxMemoryShape</code> are exclusive; do not set them both.
<code>gxMemoryShape</code>	0x0400	QuickDraw GX is to keep this shape loaded in memory as long as possible. When this attribute is set, QuickDraw GX writes this shape out to disk after all shapes are written that do not have this attribute set. The attributes <code>gxMemoryShape</code> and <code>gxDiskShape</code> are exclusive; do not set them both.

## Default Shapes

When you first create a shape of a given shape type, QuickDraw GX provides an initial value for each property; those initial values define the starting characteristics of the shape. The shape QuickDraw GX creates is a copy of the default shape for that shape type (such as a line, rectangle, or glyph). There is one default shape for each shape type. These are the default properties:

- No geometry. All applicable values and counts are set to 0.
- A shape fill that depends on the shape type:
  - The default empty shape has no fill.
  - The default point, line, and curve shapes have open-frame fill.
  - The default rectangle, polygon, path, full, bitmap, and picture shapes have even-odd fill.
  - The default text, glyph, and layout shapes have winding fill.

## Shape Objects

- A `nil` style reference, which is equivalent to a reference to the default style object. See the chapter “Style Objects” in this book for a description of the default style object. Graphic shapes all share a single common default style; each different type of typographic shape (text, glyph, and layout) uses its own default style.
- A `nil` ink reference, which is equivalent to a reference to the default ink object. See the chapter “Ink Objects” in this book for a description of the default ink object. All shapes except bitmaps share a common default ink object.
- A `nil` transform reference, which is equivalent to a reference to the default transform object. See the chapter “Transform Objects” in this book for a description of the default transform object. All shapes share a common default transform, with the exception of the picture shape, which has its own default transform.
- No attributes set (except for bitmap, picture, and layout shapes, which have the `gxMapTransformShape` attribute set).
- An owner count of 1.
- An empty tag list.

After creating the shape, you can change its characteristics to customize it; for example, you can give it a specific geometry. Or, if you want to create several shapes with the same customized characteristics, you can change the default shape itself to suit your purposes. If you do this, each shape that you create thereafter has the customized characteristics. See the section “Getting and Setting the Default Shape Objects” beginning on page 2-23, and the section “Resetting a Shape Object’s Properties to Their Default Values” beginning on page 2-31, for more information.

## Modifying Shape Properties

After you have created a shape of a given type, you can set its various properties in order to make it useful for your purposes. Some generally applicable property-setting functions, such as those that modify the attributes or owner count, are described in this chapter. Others, more specific to individual shape types, are described in the appropriate chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*.

In addition, however, QuickDraw GX provides several general-purpose functions that directly affect the geometry or type of a shape. The functions of this type described in this chapter allow you to

- copy the geometry from one shape into another, which can have the effect of changing its type (see “Copying the Geometry From One Shape to Another” beginning on page 2-29)
- directly manipulate shape geometry in QuickDraw GX memory (see “Directly Manipulating a Shape’s Geometry” beginning on page 2-34)
- convert a shape of one type, such as a rectangle, to another, such as a line or a bitmap (see “Converting Shapes From One Type to Another” beginning on page 2-32)

## Shape Objects

The functions that convert from one shape type to another are described in this chapter, but the rules for and consequences of conversion among shape types are specific to each shape type and thus are not described here. Table 2-5 on page 2-33 lists the chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography* where you can find this information.

## Drawing Shapes

---

Two of the most fundamental and common operations you perform on shapes are drawing and its complement, hit-testing (interpreting mouse clicks or otherwise relating coordinate position to shape geometry).

You can draw a shape as soon as you have created it and set its properties (and those of its related objects). The view ports listed in the transform object associated with the shape determine where the drawn shape appears. Drawing takes into account all the information in the shape's transform, ink, style, and shape objects. This chapter describes the basic drawing function that can draw any kind of shape. See "Drawing Shapes" beginning on page 2-35 for more description and an example of drawing.

## Hit-Testing Shapes

---

Hit-testing is the process of converting a point in the displayed representation of a shape to a location in the shape object's geometry. You can use hit-testing for shape selection, highlighting, or positioning the caret in text.

When you hit-test a shape, you can in most cases determine which part of a shape's geometry corresponds (within a certain *tolerance*, or distance) to the point you are testing against. For example, you can tell if a point is exactly on a line, or only close to it; and you can tell which edge of which glyph in a line of text is closest to the hit point.

QuickDraw GX provides a general hit-testing capability for all shapes, a specialized hit-test for testing picture shapes, another specialized hit-test for use with layout shapes, and another specialized hit-test for comparing shapes to specific pixels on a display device. See "Hit-Testing Shapes" beginning on page 2-36 for more information on the specific functions.

When you use the general hit-testing function, it returns one or more shape parts, which specify the parts of the shape's geometry corresponding to the hit point. The parts of a shape's geometry for which you can hit-test depend on the kind of shape. For example, for a typographic shape, the possible parts include those shown on the left side of Figure 2-5:

- bounds: the bounding rectangle enclosing the entire typographic shape
- glyph bounds: the bounding box for an individual glyph
- glyph first part: the left half of the glyph
- glyph second part: the right half of the glyph
- side bearing: the space on either side of the glyph

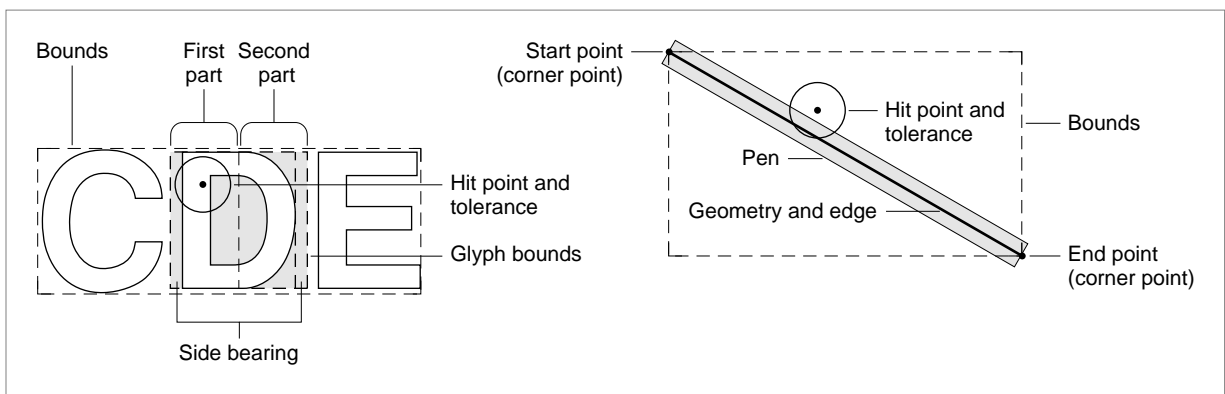


## Shape Objects

As another example, the possible parts for a line include those shown on the right side of Figure 2-5:

- bounds: the bounding rectangle enclosing the start and end points of the line
- edge: the start and end points and all the points between them on the line
- pen: a polygon with half the width of the pen on each side of the line
- geometry: the line's edge plus all area enclosed by it (in this case none, because a line encloses no area)

**Figure 2-5** Shape parts for hit-testing



The shape parts that you can test for are defined in the `gxShapeParts` enumeration, shown on page 2-37 and described in more detail in the chapter “Transform Objects” in this book. Before performing the hit-test, you set up—in the transform object of the shape you are testing—a mask structure that defines all the shape parts that you want to test for. QuickDraw GX tests only for those parts that you specify in the shape parts mask.

For example, if the hit point on the right side of Figure 2-5 is within the tolerance of the geometry part, the function will determine that it corresponds to the bounds, the geometry, the pen, and the edge. If you want to test for geometry alone, then, you could exclude all but geometry from the test. For hit-testing the text on the left side of Figure 2-5, you might be interested only in whether the hit is within the bounding rectangle of the shape and which side of which glyph it corresponds to, so you can specify the shape parts appropriately.

When you set up the hit-test parameters, you also specify a tolerance. Tolerance is a distance (in units of geometry space), and it defines a circular area centered on the hit point. Any part that falls within that area is considered to correspond to the hit point.

## Saving and Restoring Shapes

---

In memory, a QuickDraw GX shape consists of a shape object and (by reference) several other objects, including a style, an ink, and a transform. The locations and internal structures of those objects are private.

If you need to save a shape in a document or other external storage form, or transmit it across a network, or otherwise preserve its information in a public format, you can convert, or *flatten*, its object-based description into a stream-based description. Conversely, you can restore the object-based description of an object from its flattened form.

The flattened, stream-based format for most objects is documented in the stream format chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. (Fonts have their own flattened format; see the font objects chapter of *Inside Macintosh: QuickDraw GX Typography* for more information.) How to flatten and unflatten shape objects is described in the section “Flattening and Unflattening Shapes” beginning on page 2-39 in this chapter.

## Using Shape Objects

---

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

This section describes how you can

- create and manipulate shape objects
- manipulate shape object properties
- convert shapes from one type to another
- directly manipulate shape geometry
- flatten and unflatten shapes
- draw and hit-test shapes

## Creating and Manipulating Shape Objects

---

This section describes how you can create and interact with shape objects as whole entities—to create, dispose of, copy, compare, clone, cache, load, and unload them. It also describes how to manipulate the default shapes. Manipulating the *properties* of shapes is described under “Manipulating Shape Object Properties” beginning on page 2-28.

## Getting and Setting the Default Shape Objects

QuickDraw GX defines a default shape object for each shape type. These defaults are the templates QuickDraw GX uses when creating new shape objects, and you can change them to suit your purposes. Note, however, that changing the geometry for a default shape has no effect when subsequent shapes are created from the default one. A newly created shape never contains a geometry.

You can use the `GXGetDefaultShape` function to examine one of the default shape objects and the `GXSetDefaultShape` function to replace one of the default shape objects.

The properties common to all default shape objects are described under “Default Shapes” on page 2-18. Default properties specific to graphic or typographic shapes are described in *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*, respectively.

The following code fragment uses `GXGetDefaultShape` to change the characteristics of the ink object referenced by the default line shape. The code obtains a reference to the default shape, and creates a temporary ink reference (`tempInk`) to the shape’s ink object. It changes the temporary ink’s color and transfer mode (with library functions `SetInkCommonColor` and `SetInkCommonTransfer`), and then assigns the modified ink back to the default shape:

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

The code disposes of the temporary ink after assigning it to the default shape, because that temporary reference is no longer needed.

### Note

If you have created a shape object, and want to restore some of its default values, you can use the `GXResetShape` function. See the section “Resetting a Shape Object’s Properties to Their Default Values” beginning on page 2-31. ♦

The `GXGetDefaultShape` function is described on page 2-52. The `GXSetDefaultShape` function is described on page 2-53.

## Shape Objects

## Creating and Disposing of Shape Objects

---

QuickDraw GX provides a number of ways for you to create a new shape object. This section describes the `GXNewShape` function, which creates a copy of the default shape for the shape type you specify. You can then customize the shape using the techniques described in the section “Manipulating Shape Object Properties” beginning on page 2-28. Other ways to create and customize specific types of shape objects are described in the chapters that describe shapes in *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*. Note that you can also create a new shape by copying an existing one: see the section “Copying, Comparing, and Cloning Shape Objects” beginning on page 2-25.

Before you can create a shape or any other object, you need to be in the QuickDraw GX environment. You are not required to make any calls to accomplish this, however; QuickDraw GX sets up the environment for your application when you make your first QuickDraw GX call. If you nevertheless wish to control your application’s memory use in the QuickDraw GX environment, you can use the functions `GXNewGraphicsClient` and `GXEnterGraphics`, described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

The following code fragment creates a rectangle shape (`rectShape`), assigns it a fill type (closed-frame fill), and assigns its ink object a gray color (using the library function `SetShapeCommonColor`):

```
rectShape = GXNewShape (gxRectangleType);
GXSetShapeFill (rectShape, gxClosedFrameFill);
SetShapeCommonColor (rectShape, gxGray);
```

The following code fragment creates a picture shape (`docPage`) to represent the page of a document that is to be printed. It sets the `gxUniqueItemsShape` shape attribute to make sure each item in the picture has a unique reference:

```
docPage = GXNewShape(gxPictureType);
GXSetShapeAttributes(docPage, gxUniqueItemsShape);
```

(Note that this method of assigning an attribute clears all other attributes, which may be undesirable. In general, you would first call `GXGetShapeAttributes`, modify the returned attributes as needed, and then call `GXSetShapeAttributes` to reassign them.)

To delete your application’s reference to a shape object, call the `GXDisposeShape` function. You must be sure to dispose of every shape that you create. For the `docPage` shape you would make this call:

```
GXDisposeShape (docPage);
```

## Shape Objects

Note that calling `GXDisposeShape` for a particular shape object may or may not actually release the memory allocated for that object, depending on its owner count. `GXDisposeShape` decreases the shape object's owner count by 1; if that brings the owner count to 0, the shape is completely deleted and its memory released (and the owner count of each object that the shape object references is then decremented). See "Manipulating a Shape Object's Owner Count" on page 2-31.

The `GXNewShape` function is described on page 2-54. The `GXDisposeShape` function is described on page 2-55.

## Getting the Size of a Shape Object in Memory

---

Although the sizes of style, ink, and transform objects are relatively constant, shape objects vary greatly in size, mostly due to the differences in their geometries. The `GXGetShapeSize` function allows you to find out how much memory a shape occupies.

The `GXGetShapeSize` function returns only the amount of memory currently being used to represent the shape. Because QuickDraw GX can automatically unload objects from memory, the size returned by `GXGetShapeSize` does not accurately reflect the size of the object if it has been unloaded. You can call the `GXLoadShape` function before calling `GXGetShapeSize` to get a more accurate size, if necessary.

The `GXGetShapeSize` function is described on page 2-56.

## Copying, Comparing, and Cloning Shape Objects

---

You can use the `GXCopyToShape` and `GXCopyDeepToShape` functions to copy all of the information from one shape to another or to create a new copy of a shape. The two functions are identical except that `GXCopyDeepToShape` copies more information for these shape types: for bitmap shapes, it also copies the pixel image; for picture shapes, it makes a new copy of each shape in the picture; and for glyph and layout shapes, it copies the style list.

The following code fragment copies a shape to make a version having special visual characteristics. It makes a temporary shape (`tempTextShape`) that is a copy of a text shape (`textShapeFromPicture`) within a picture shape representing a document page. The `GXCopyDeepToShape` function is not needed in this case because a text shape, unlike a glyph shape or layout shape, cannot have a style list to copy. The code doubles the size of the text and moves it by 100 points vertically before inserting it back into the page and disposing of the temporary reference.

## Shape Objects

Note that this code makes use of the QuickDraw GX `ff` macro, a shorthand version of the `IntToFixed` macro. Both functions are described in the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

```
GXGetPictureParts(thePage, 2, 1, &textShapeFromPicture,
                  nil, nil, nil);
tempTextShape = GXCopyToShape (nil, textShapeFromPicture);

GXScaleShape(tempTextShape, ff(2), ff(2), 0, 0);
GXMoveShape(tempTextShape, 0, ff(100));

GXSetPictureParts(thePage, 3, 0, 1, &tempTextShape,
                  nil, nil, nil);
GXDisposeShape(tempTextShape);
```

You can test if two shape references refer to the same shape object by simply testing the references for equality. You can also compare two different shape objects for equality with the `GXEqualShape` function. For two shapes to be equal, their fill properties must be equal and their geometries must be identical. See the `GXEqualInk`, `GXEqualStyle`, and `GXEqualTransform` function descriptions in the chapters “Ink Objects,” “Style Objects,” and “Transform Objects,” respectively, in this book for the requirements for equality. Shape copies created by `GXCopyToShape` or `GXCopyDeepToShape` are always equal to the shape from which they were copied.

**Equivalent geometries are not identical**

Some shapes have equivalent, but not identical, geometries, and are thus not considered equal by `GXEqualShape`. For example, two polygons might have identical geometries, except that one has a duplicate point at one of its corners. The shapes are equivalent in form, but their geometries are not identical. You can remove such duplicate points with the `GXReduceShape` function, described in the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*. ♦

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

Functionally, `GXCloneShape` does nothing more than increase the owner count of a shape object. For more information about cloning objects, see the chapter “Introduction to Objects” in this book. For information on manipulating shape owner counts, see the section “Manipulating a Shape Object’s Owner Count” beginning on page 2-31 of this chapter.

The `GXCopyToShape` function is described on page 2-57. The `GXCopyDeepToShape` function is described on page 2-58. The `GXEqualShape` function is described on page 2-60. The `GXCloneShape` function is described on page 2-61.

## Caching Shape Objects

---

Before QuickDraw GX draws any shape, it first performs some preliminary calculations on the shape's data (such as finding the shape's bounds) and stores the information in a shape cache.

In certain circumstances, you can improve the way drawing occurs on the screen by requesting that QuickDraw GX create the caches before you actually draw the shapes. For example, if you are drawing many shapes at once, you can cache all of the shapes before you draw any of them. In this way, you can minimize the amount of time between the appearance of the first shape and the completion of the last shape.

You can use the `GXCacheShape` function to create caches before drawing and you can use the `GXDisposeShapeCache` function to release the memory held by a shape cache. The `GXGetShapeCacheSize` function returns information about the size of the cache in memory.

The `GXCacheShape` function works somewhat differently from the `gxCachedShape` attribute (see Table 2-4 on page 2-16). Setting the `gxCachedShape` attribute causes QuickDraw GX to cache and predraw a shape into a compressed offscreen bitmap the first time it is drawn. Then, when you call `GXDrawShape`, the predrawn shape is simply transferred to the screen. Setting the `gxCachedShape` attribute causes very fast drawing but may greatly increase the memory required to store a shape, especially for large shapes. Calling `GXCacheShape` does not increase the memory required to draw a shape. For the fastest possible drawing (but the slowest preparation for drawing), set the `gxCachedShape` attribute and also call `GXCacheShape` before drawing.

You are not required to use any of the functions in this section. QuickDraw GX automatically creates shape caches when you draw a shape and automatically deletes shape caches when memory is low. You only need to use these functions when you want to improve your application's drawing speed.

The `GXCacheShape` function is described on page 2-62; The `GXDisposeShapeCache` function is described on page 2-63; The `GXGetShapeCacheSize` function is described on page 2-64.

## Loading and Unloading Shape Objects

---

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

You call the `GXLoadShape` function to make sure that a shape object is in memory; if necessary, QuickDraw GX brings the object into memory from an unloaded state. You can call the `GXUnloadShape` function to instruct QuickDraw GX that it is free to unload the shape object at any time.

## Shape Objects

Rather than explicitly instructing QuickDraw GX to load or unload an object, you can also set either the `gxDiskShape` or the `gxMemoryShape` attribute for the shape, which permanently affects the priority with which QuickDraw GX loads or unloads the shape. Shape attributes are described in Table 2-4 on page 2-16.

The `GXLoadShape` and `GXUnloadShape` functions are described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## Manipulating Shape Object Properties

---

This section describes how to manipulate the properties of shape objects, including those that are references to other objects. In most cases, a pair of functions respectively get and set a property. You call the `GXGetShapeProperty` function to get a copy of the shape property you need; you call the `GXSetShapeProperty` function to assign a value to a property.

For manipulating shape objects as a whole, see “Creating and Manipulating Shape Objects” beginning on page 2-22.

## Getting and Setting a Shape Object’s Type, Fill, and Attributes

---

The functions described in this section get and set shape properties that are numerical values.

You can use the `GXGetShapeType` function to find the shape type of an existing shape, and the `GXSetShapeType` function to convert an existing shape from one shape type to another. The section “Converting Shapes From One Type to Another” beginning on page 2-32 summarizes the kinds of shape conversions QuickDraw GX supports. Beyond that section and the descriptions in Table 2-1 on page 2-9, this book does not discuss specific shape types. See *Inside Macintosh: QuickDraw GX Typography* for more information on the typographic shape types—text, glyph, and layout. (Note that `GXSetShapeType` even allows you to convert typographic shapes to graphic shapes of certain types.) See *Inside Macintosh: QuickDraw GX Graphics* for more information on graphic shape types.

The following code fragment determines the number of items (`numParts`) in a picture shape (`theShape`). The code uses `GXGetShapeType` to screen out any shape that is not a picture shape:

```
typeOfShape = GXGetShapeType(theShape);
if (typeOfShape == gxPictureType)
    numParts = GXGetPicture(theShape, nil, nil, nil, nil);
```

You can use the `GXGetShapeFill` function to find the fill of an existing shape, and the `GXSetShapeFill` function to set the fill of a shape when you create or modify it. Beyond the descriptions in Table 2-2 on page 2-13, this book does not discuss specific shape fills. See *Inside Macintosh: QuickDraw GX Typography* and *Inside Macintosh: QuickDraw GX Graphics* for more information on the valid typographic and graphic shape fills.



## Shape Objects

You can use the `GXGetShapeAttributes` function to find the attributes of an existing shape and the `GXSetShapeAttributes` function to set the attributes of a shape. Shape attributes are described in the section “Shape Attributes” beginning on page 2-16.

The following code fragment is a drawing loop that rotates a text shape (`theText`) six times around the point (`x`, `y`) by 15 degrees each time, and adds the shape to a picture (`gthePage`) after each rotation. (It also changes the color at each rotation, for better visibility of the overlapping text.) The loop sets the `gxMapTransformShape` attribute of the shape, which assures that the shape geometry itself is not affected by the rotation, and thus there is no loss of precision in the geometry with repeated rotations:

```
GXSetShapeAttributes(theText, gxMapTransformShape);
for (loop = 0; loop < 6; loop++)
{
    GXSetShapeColor(theText, &textColor);
    GXRotateShape(theText, ff(15), x, y);
    GXSetPictureParts(gthePage, 0, 0, 1, &theText, nil, nil, nil);
    textColor.element.hsv.hue += 0x0940;
}
```

Note that the `gxUniqueItemsShape` attribute of `gthePage` must be set for this to work.

You can use `GXGetShapeAttributes` in combination with the `GXSetShapeAttributes` function to set and clear single attribute flags. For example, to clear the `gxDiskShape` attribute of a shape referenced by the variable `target`, you could use the following code:

```
GXSetShapeAttributes(target,
    GXGetShapeAttributes(target) & ~gxDiskShape);
```

Conversely, to set the `gxDiskShape` attribute, you could use the following code:

```
GXSetShapeAttributes(target,
    GXGetShapeAttributes(target) | gxDiskShape);
```

The `GXGetShapeType` function is described on page 2-66. The `GXSetShapeType` function is described on page 2-66. The `GXGetShapeFill` function is described on page 2-68. The `GXSetShapeFill` function is described on page 2-69. The `GXGetShapeAttributes` function is described on page 2-74. The `GXSetShapeAttributes` function is described on page 2-74.

## Copying the Geometry From One Shape to Another

Like type, fill, and attributes, geometry is a property of a shape object. However, you access and manipulate a shape’s geometry somewhat differently from other properties.

## Shape Objects

The `GXSetShapeGeometry` function copies the geometry (and the shape type, if the shapes are of different types) from one shape object into another. To make the function call requires two object references, and no reference to or specification of either object's geometry. There is no associated `GXGetShapeGeometry` call. Using `GXSetShapeGeometry` is a simple way to reuse an existing shape by turning it into a copy of another shape. As with `GXSetShapeType`, this book does not discuss the specific rules for and consequences of converting one shape type to another with `GXSetShapeGeometry`. See *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography* for conversion information for graphic and typographic shape types.

To do more than simply copy geometries—to gain access to and actually manipulate the contents of a shape's geometry—requires another set of functions, including the `GXGetShapeStructure` function. See the section “Directly Manipulating a Shape's Geometry” beginning on page 2-34. In most situations, however, you use functions specific to a given shape type to manipulate that type of shape's geometry. Those kinds of functions are described, along with each shape type, in *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*.

To copy an entire object, rather than just its geometry, you can use the `GXCopyToShape` or `GXCopyDeepToShape` functions; see “Copying, Comparing, and Cloning Shape Objects” on page 2-25.

The `GXSetShapeGeometry` function is described on page 2-67.

## Getting and Setting a Shape Object's Style, Ink, and Transform

---

Every QuickDraw GX shape object has an associated style object, ink object, and transform object. You can use the `GXGetShapeStyle`, `GXGetShapeInk`, and `GXGetShapeTransform` functions to determine which of each type of object is referenced by a particular shape. Conversely, you can use the `GXSetShapeStyle`, `GXSetShapeInk`, and `GXSetShapeTransform` functions to change these references.

Because style objects can be shared among different QuickDraw GX shapes, the `GXGetShapeStyle` function can return a reference to the same style object for two different shapes. Likewise, the `GXGetShapeInk` and `GXGetShapeTransform` functions can return identical ink objects or transform objects for different shapes.

Calling `GXSetShapeStyle`, `GXSetShapeInk`, or `GXSetShapeTransform` increments the owner count of the specified style, ink, or transform object by 1, and disposes of the previously assigned style, ink, or transform. In certain cases, depending on how you create such an object or assign it to a shape, you may need to modify that object's owner count explicitly; see “Manipulating a Shape Object's Owner Count” on page 2-31.

The following code fragment draws a dashed version of a shape. The code first calls `GXGetShapeStyle` to obtain the style object attached to the shape `theShape`; it then clones the style and assigns a temporary reference (`saveStyle`) to the style. The code then assigns different style properties to the shape and draws it. After drawing the shape, the code restores the original style to the shape, using `GXSetShapeStyle`:

## Shape Objects

```

saveStyle = GXCloneStyle(GXGetShapeStyle(theShape));
GXSetShapePen(theShape, ff(1));
GXSetShapeDash(theShape, &dash);
GXDrawShape(theShape);

GXSetShapeStyle(theShape, saveStyle);
GXDisposeStyle(saveStyle);

```

As usual, after it is finished with the temporary reference `saveStyle`, the code disposes of it. For more information and examples of cloning, see for example the discussions of owner count in the chapter “Style Objects” in this book.

The `GXGetShapeStyle` function is described on page 2-69; the `GXSetShapeStyle` function is described on page 2-70. The `GXGetShapeInk` function is described on page 2-71; the `GXSetShapeInk` function is described on page 2-71. The `GXGetShapeTransform` function is described on page 2-72; the `GXSetShapeTransform` function is described on page 2-73.

### Resetting a Shape Object's Properties to Their Default Values

When you create a new shape with the `GXNewShape` function, QuickDraw GX creates the new shape object by copying the appropriate default shape object. QuickDraw GX does not create a new style, ink, or transform object for the new shape, however. Instead, the new shape contains references to the same style, ink, and transform as the corresponding default shape. You are free to install a new style, ink, or transform in the shape using functions such as `GXSetShapeStyle`, `GXSetShapeInk`, and `GXSetShapeTransform`.

If you do install a new style, ink, or transform in a shape and you want to revert back to the default style, ink, and transform, you can use the `GXResetShape` function. This function also resets the shape's attributes and fill properties to match the default shape, but does not alter the shape's geometry, owner count, or tag list.

The `GXResetShape` function is described on page 2-75.

### Manipulating a Shape Object's Owner Count

The owner count of an object indicates the number of current references to that object. In general, QuickDraw GX manages owner counts for you. For example, when you create a new shape object you give it a variable name such as `myShape`. QuickDraw GX sets the owner count of the new shape to 1, because your application variable is the only current reference to the shape. As another example, when you add a shape to a picture, QuickDraw GX increments the shape's owner count, corresponding to the new reference to the shape contained in the picture.

## Shape Objects

The following code fragment is part of a routine that constructs a house image (`gOurHouse`) as a picture shape, building it out of individual geometric shapes. As each component shape (`houseBorderShape` and `doorShape`, in this fragment) is added to the picture shape, its owner count is increased; to balance that increase, and because that component shape's reference is no longer needed, it is disposed of.

```
GXSetShapeFill(houseBorderShape, gxHollowFill);
GXSetPictureParts(gOurHouse, 1, 0, 1, houseBorderShape,
                  nil, nil, nil);
GXDisposeShape(houseBorderShape);

GXSetShapeFill(doorShape, gxHollowFill);
GXSetPictureParts(gOurHouse, 1, 0, 1, doorShape,
                  nil, nil, nil);
GXDisposeShape(doorShape);
```

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

The `GXGetShapeOwners` function is described on page 2-76. The `GXCloneShape` function is described on page 2-61. The `GXDisposeShape` function is described on page 2-55.

## Getting and Setting a Shape Object's Tag References

---

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

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

The `GXGetShapeTags` function is described on page 2-77. The `GXSetShapeTags` function is described on page 2-78.

## Converting Shapes From One Type to Another

---

QuickDraw GX allows you to change the types of the shape objects you have created. You use the `GXGetShapeType` function, described on page 2-66 of this chapter, to determine the type of a shape. To convert a shape to a new type, you use the `GXSetShapeType` function, described on page 2-66 of this chapter.

## Shape Objects

The rules for conversion among shape geometries are specific to each shape type and thus are not described here. See the appropriate chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography* for this information. Table 2-5 describes where to look in each book for information regarding each possible kind of conversion.

**Table 2-5** Where to find information on shape-type conversion

	To a geometric shape	To a bitmap shape	To a picture shape	To a typographic shape
<b>From a geometric shape</b>	See “Geometric Shapes” in <i>QuickDraw GX Graphics</i>	See “Bitmap Shapes” in <i>QuickDraw GX Graphics</i>	See “Picture Shapes” in <i>QuickDraw GX Graphics</i>	(not possible)
<b>From a bitmap shape</b>	See “Geometric Shapes” in <i>QuickDraw GX Graphics</i>	(no change)	See “Picture Shapes” in <i>QuickDraw GX Graphics</i>	(not possible)
<b>From a picture shape</b>	See “Geometric Shapes” in <i>QuickDraw GX Graphics</i>	See “Bitmap Shapes” in <i>QuickDraw GX Graphics</i>	(no change)	(not possible)
<b>From a typographic shape</b>	See “Typographic Shapes” in <i>QuickDraw GX Typography</i>	See “Bitmap Shapes” in <i>QuickDraw GX Graphics</i>	See “Picture Shapes” in <i>QuickDraw GX Graphics</i>	See “Typographic Shapes” in <i>QuickDraw GX Typography</i>

Another common kind of shape conversion is not from one shape type to another, but from standard object form into primitive form. Some functions, such as `GXSetShapeClip`, described in the chapter “Transform Objects” in this book, require a primitive shape to hold the clip shape. A **primitive shape** is a shape whose stylistic information has been incorporated into the shape’s geometry. For example, a horizontal line with a thick pen style becomes a rectangle when converted to a primitive shape. To make a shape into a clip, you first convert it to its primitive form with the function `GXPrimitiveShape`. For more information about primitive shapes in general, see the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*. For information on primitive shapes for typographic shapes, and the difference between using `GXPrimitiveShape` and `GXSetShapeType` to obtain a primitive shape, see the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*.

## Shape Objects

## Directly Manipulating a Shape's Geometry

---

The geometry of a shape object is its most central property. Unlike other properties, it can be made accessible to you as a structure that you can modify directly, in place in QuickDraw GX memory. QuickDraw GX provides a group of functions with which you can access a shape's geometry and notify QuickDraw GX once you have modified it. Note that in most cases you don't need to do this; QuickDraw GX provides many functions, specific to each type of shape, with which you can access and modify geometry. The functions described here are provided as an added convenience.

These functions do not provide you with information about the formats of the data structures that make up shape geometries; they simply give you a pointer to the geometry. How you manipulate that information depends on the type of shape whose geometry you are accessing. The structures of individual shape geometries are described in the shape-specific chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*.

Before accessing a shape's geometry, you must set its `gxDirectShape` attribute to make sure that it is loaded into directly accessible memory. To access the geometry, you first call the `GXLockShape` function to make sure the shape object doesn't move until you are finished with it. You then call the `GXGetShapeStructure` function, which returns a pointer to the shape's geometry. You can then modify the geometry as needed. Once finished, you call `GXUnlockShape` to free the shape object for relocation in memory as needed. Finally, you must call `GXChangedShape` to notify QuickDraw GX that you have changed the geometry.

Listing 2-1 is a partial listing of a function that accesses the geometry of the path shape `myShape`, manipulating its geometry as a `gxPaths` structure in a buffer of size `size`.

---

**Listing 2-1**      Directly accessing a shape's geometry

```
.
. /* set up the shape (not shown) */
.
/* set the direct shape attribute if not set */
GXSetShapeAttributes (myShape,
                     GXGetShapeAttributes(myShape) | gxDirectShape);

/* lock and examine or change the shape */
GXLockShape(myShape);
shapeStruct = (gxPaths*)GXGetShapeStructure(myShape, &size);
.
. /* unlock the shape as soon as access no longer needed */
.
GXUnlockShape(myShape);

/* notify QuickDraw GX of a change only if geometry changed */
GXChangedShape(myShape);
```

## Shape Objects

**IMPORTANT**

Memory-handling complications can occur with locked objects. Locking an object fragments the QuickDraw GX heap, which can result in lower performance. Furthermore, if a fragmented-memory condition occurs during a call, QuickDraw GX may unlock all objects and restart the call. Therefore, be careful about performing memory-intensive operations while there are locked objects in QuickDraw GX memory; they may become unlocked and be moved. ▲

The GXLlockShape function is described on page 2-80. The GXGetShapeStructure function is described on page 2-82. The GXUnlockShape function is described on page 2-81. The GXChangedShape function is described on page 2-83.

## Drawing and Hit-Testing Shapes

Drawing and hit-testing are common actions you may perform with any kind of shape. The most basic QuickDraw GX drawing function is GXDrawShape, although there are other functions for drawing specific types of shapes. Only GXDrawShape is described here.

The functions you use for hit-testing are GXHitTestShape, GXHitTestPicture, GXHitTestLayout, and GXHitTestDevice. Only GXHitTestShape is described here.

## Drawing Shapes

Drawing a shape is the logical conclusion to creating it and setting its properties. Drawing occurs in the view port or view ports specified in the transform object associated with the shape. Drawing takes into account all the information in the shape's transform, ink, style, and shape objects.

What it means to draw a specific type of shape and how changing the information in a shape alters its drawn appearance is described, along with each type of shape, in *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*. Furthermore, for many shape types, QuickDraw GX provides specialized drawing functions, such as GXDrawLine and GXDrawGlyphs—described in those books—that allow you to create, draw, and dispose of an object with a single call.

At its most basic, though, creating and drawing a shape is as simple as the following listing for creating and drawing a path shape shows:

```
gxShape myShape;                                /* allocate the variable */
myShape = GXNewShape(gxPathType);                /* create the shape */
.
.                                                /* set its properties */
.
GXDrawShape(myShape);                            /* draw it */
```

The GXDrawShape function is described on page 2-84.

## Shape Objects

## Hit-Testing Shapes

---

Hit-testing converts a coordinate location to a shape-geometry location. It can give you feedback on user actions involving a shape you have drawn. For example, you use hit-testing to select a shape the user has clicked the mouse over, to select a point within a shape, or to position the insertion point and draw the caret within the text of a typographic shape.

QuickDraw GX provides a general hit-testing function for all shapes, plus specialized functions for hit-testing picture shapes, layout shapes, and pixels on a display device:

- `GXHitTestShape` tests a point in local space against a shape's geometry. The test tells you which part of a shape's geometry—out of a specified set of parts—corresponds (within the tolerance) to the point you are testing with. The `GXHitTestShape` function is described in this chapter.
- `GXHitTestPicture` tests a point in local space against a picture shape. The test tells you which part of which shape within the picture corresponds (within the tolerance) to the point you are testing against (subject to the constraints on shape overlap and hierarchy that you provide). The `GXHitTestPicture` function is described in the picture shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.
- `GXHitTestLayout` tests a point in local space against the text of a layout shape. The test tells you, along with other information, which character in the text corresponds to the point. Note that you use `GXHitTestShape` to test typographic shapes other than layout shapes, and you can use it for layout shapes also; it gives different kinds of information from `GXHitTestLayout`. The `GXHitTestLayout` function is described in the layout carets, highlighting, and hit-testing chapter of *Inside Macintosh: QuickDraw GX Typography*.
- `GXHitTestDevice` tests a pixel (a point in device space) against a shape's geometry. The test tells you whether or not any part of the shape's geometry is within a certain distance of the pixel. The `GXHitTestDevice` function is described in the chapter "View-Related Objects" in this book.

When you hit-test a shape with `GXHitTestShape`, you must first set up the shape parts mask and the tolerance, two components of the **hit-test parameters** property of a shape's transform object. You pass that information to `GXHitTestShape`, and QuickDraw GX returns information in the **hit-test info structure**.

The tolerance is a distance (in units of geometry space), and it defines a circular area centered on the hit point. Any part that falls within that area is considered to correspond to the hit point.

## Shape Parts

---

When you use `GXHitTestShape`, it returns one or more shape parts, which specify the parts of the shape's geometry corresponding to the hit point. The parts of a shape's geometry for which you can hit-test depend on the kind of shape. The shape parts that you can test for are defined in the `gxShapeParts` enumeration. Before calling `GXHitTestShape`, you set up, in the transform object, a mask of all the shape parts that you want to test for. `GXHitTestShape` can test only for parts that you specify in the shape parts mask. These are the possible values to put into the mask:



## Shape Objects

```
enum gxShapeParts {          /* (in order of evaluation) */
    gxNoPart                = 0,
    gxBoundsPart            = 0x0001,
    gxGeometryPart          = 0x0002,
    gxPenPart                = 0x0004,
    gxCornerPointPart        = 0x0008,
    gxControlPointPart       = 0x0010,
    gxEdgePart               = 0x0020,
    gxJoinPart               = 0x0040,
    gxStartCapPart           = 0x0080,
    gxEndCapPart             = 0x0100,
    gxDashPart               = 0x0200,
    gxPatternPart            = 0x0400,
    gxGlyphBoundsPart        = gxJoinPart,
    gxGlyphFirstPart         = gxStartCapPart,
    gxGlyphLastPart          = gxEndCapPart,
    gxSideBearingPart        = gxDashPart,
    gxAnyPart                = gxBoundsPart | gxGeometryPart |
        gxPenPart | gxCornerPointPart | gxControlPointPart |
        gxEdgePart | gxJoinPart | gxStartCapPart |
        gxEndCapPart | gxDashPart | gxPatternPart
};

typedef long gxShapePart;
```

These values are described in more detail in the chapter “Transform Objects” in this book. Note that values specifying join, cap, and dash parts in geometric shapes are used in typographic shapes to specify various glyph parts. Note also that you can specify no parts or all parts in the mask. You decide which shape parts are appropriate for your needs.

**Hit-Test Info Structure**

When you call `GXHitTestShape`, it returns some information as a function result and other information in a hit-test info structure. The first three fields of the hit-test info structure give all the relevant information about the hit:

```
struct gxHitTestInfo {
    gxShapePart    what;
    long           index;
    Fixed          distance;
    gxShape        which;
    gxShape        containerPicture;
    long           containerIndex;
    long           totalIndex;
};
```

## Shape Objects

The `what` field tells you which shape parts out of those specified in your mask were hit, if any. It is identical to the `GXHitTestShape` function result.

The `index` field tells you the index number of the point in the geometry that is closest to the hit point.

The `distance` field tells you how far, in geometry coordinates, the hit point is from the first shape part that was hit. `GXHitTestShape` analyzes shape parts in a specific order—the order listed in the `gxShapeParts` enumeration. By carefully specifying shape parts, you can use `GXHitTestShape` to obtain specific distance information for a given part. For example, if you are hit-testing a line like that shown in Figure 2-5 on page 2-21, you can determine the distance from the hit point to the pen if you exclude both bounds and geometry from the test.

The remaining fields in the hit-test info structure are not used by `GXHitTestShape`.

### Hit-Testing Example

---

Listing 2-2 uses hit-testing to determine whether a point (`aPoint`) is contained in the geometry that represents a shape (`gShape`). The code sets up a shape-part mask (`mask`) specifying that only the geometry it to be tested for, and calls the `GXSetShapeHitTest` function to assign the mask, plus a tolerance of zero, to the shape's transform.

---

**Listing 2-2**      Hit-testing a line

```
gxShape      pointShape;
gxPoint      aPoint = {ff(50), ff(51)};
gxShapePart  mask = gxGeometryPart;
gxShapePart  resultMask;
gxHitTestInfo resultInfo;

pointShape = GXNewPoint(&aPoint);
GXSetShapeHitTest(gShape, mask, ff(0));
resultMask = GXHitTestShape(gShape, &aPoint, &resultInfo);
GXDisposeShape(pointShape);
```

The function result from `GXHitTestShape` tells which part of the shape was hit. Because only one part (`gxGeometryPart`) is specified and tolerance is 0, a successful hit is possible only if `aPoint` is actually within the geometry of the shape.

In the event of a successful hit, `GXHitTestShape` also fills in a `gxHitTestInfo` structure (`resultInfo` parameter) that contains additional information about the hit.

The `gxHitTestInfo` structure is described on page 2-50. The `GXHitTestShape` function is described on page 2-86. Because the shape parts to test against are specified in a shape's transform object, the list of defined QuickDraw GX shape parts, and the `GXSetShapeHitTest` function, are described in the chapter "Transform Objects" in this book.

## Flattening and Unflattening Shapes

In order to save a QuickDraw GX shape (shape object plus its referenced objects) to external storage, transmit it across a network, or save it to the Clipboard, you must convert it into an equivalent flattened, rather than object-based, description. The flattened information is a compressed and stream-based description with a public format so that applications can share the data and reconstruct the objects.

You can use the `GXFlattenShape` function to convert any shape (even a picture shape, which contains other shapes) into its flattened form. You can then store the data, examine it, or manipulate it as you wish; the data follows the format defined in the stream format chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To reconstruct a shape's object-based description from its flattened stream, you can manually create and initialize a set of objects based on the information in the stream, but if QuickDraw GX is available, it is far easier and more efficient to use the `GXUnflattenShape` function to do it automatically.

To use the flattening or unflattening functions, you first allocate a structure called a **spool block**. The spool block contains needed information and points to a buffer that holds the flattened data. In the spool block, you are required to provide a pointer to a callback spool function that you provide. The spool function reads the stream data into the buffer or writes it to a file from the buffer.

Listing 2-3 is a library function that flattens a shape and returns a handle to the flattened data. It uses a spool-block structure (`spool`) embedded within a library-defined structure (`block`) of type `UserSpool`. The function sets up the spool-block structure, including placing into it a pointer to the spool function. It specifies `nil` for the buffer pointer and 0 for the buffer size, in which case QuickDraw GX allocates a default buffer for the task. When it calls `GXFlattenShape`, the function sets two flatten flags, so that both a list of fonts and a list of all the individual glyphs used is attached to the flattened shape.

**Listing 2-3**      Flattening a shape

```
Handle ShapeToHandle(gxShape source)
{
    UserSpool block;

    block.spool.spoolProcedure = (long (*)(gxSpoolCommand,
                                         struct gxSpoolBlock *)) HandleSpoolProc;
    block.spool.buffer = nil;
    block.spool.bufferSize = 0;
    GXFlattenShape(source,
                   gxFontListFlatten | gxFontGlyphsFlatten,
                   &block.spool);
    return block.data;
}
```

## Shape Objects

Listing 2-4 is a library function that unflattens a shape from data referenced by a handle (*source*). Like Listing 2-3, it sets up a spool-block structure and places into it a pointer to the spool function. When it calls `GXUnflattenShape`, the function specifies the size of the flattened data and the list of view ports to be assigned to the unflattened shape's transform object.

---

**Listing 2-4**      Unflattening a shape

```
gxShape HandleToShape(Handle source, long count,
                      const gxViewPort portList[])
{
    UserSpool block;

    block.spool.spoolProcedure = (long (*)(gxSpoolCommand,
                                           struct gxSpoolBlock *)) HandleSpoolProc;
    block.spool.buffer = nil;
    block.spool.bufferSize = 0;
    block.data = source;
    return GXUnflattenShape(&block.spool, count, portList);
}
```

Your flattening/unflattening spool function responds to five commands from QuickDraw GX (described on page 2-92). In most cases it simply reads or writes a buffer of data at a time during the flattening or unflattening operation, and then closes up when the operation is finished. However, for special purposes you can write a spool function that parses the stream of data by reading information in the spool block and manipulating the size of the buffer that QuickDraw GX can read from or write into.

Listing 2-5 is a partial listing that shows the overall structure of a typical spool function for flattening and unflattening. This function, however, parses the stream as it is being flattened or unflattened. In the case of writing (flattening), the listing shows that the function sets the buffer size to equal the current operation size so that no more than a single operation can be flattened at once. Therefore, each time it is called, the spool file can read the fields of the spool block to determine the kind of information the current operation consists of and decide how large to make the buffer for the next write.

**Listing 2-5** A spool function that parses shape data

```

static long MyParseSpoolProc(spoolCommand command,
                             gxSpoolBlock *block)
{
    switch (command) {
        case openReadSpool:
            .
            . /* spool function prepares for unflattening */
            .
            break;
        case openWriteSpool:
            .
            . /* spool function prepares for flattening */
            .
            break;
        case closeSpool:
            .
            . /* spool function closes up when finished */
            .
            break;
        case readSpool:
            .
            . /* spool function parses and reads for unflattening */
            .
            break;
        case writeSpool:

            /* see if current operation < 32K (real buffer size) */
            if (block->spool.operationSize < 32768)

                /* set buffer size to operation size */
                block->spool.bufferSize = block->spool.operationSize;
            else
                block->spool.bufferSize = 32000; /* don't overflow */
            . /*
            .   Spool function examines spool block, parses data,
            .   writes flattened data to disk
            . */
            break;
    }
}

```

## Shape Objects

The application sets up the conditions for this spool function by first allocating a 32 KB buffer, but setting the `size` field of the spool block to 1. This causes `GXFlattenShape` or `GXUnflattenShape` to read only a single byte into the buffer the first time through, after which the spool function can analyze that byte and proceed with parsing. (For simple reading or writing, your application typically sets the `size` field to the actual size of the buffer—32 KB in this case—and the spool function does not parse the stream at all).

The `GXFlattenShape` function is described on page 2-88. The `GXUnflattenShape` function is described on page 2-90. The spool block structure is described on page 2-49. The application-defined spool function is described on page 2-91. The flatten flags are described on page 2-48.

## Shape-Related Functions Described Elsewhere

---

Table 2-6 lists every QuickDraw GX function whose name contains the word *Shape*, but whose description is not found in this chapter. For each book and chapter, the table lists the shape-related functions described in that chapter. Table 2-6 is intended to help you locate the descriptions of functions you may have been searching for in this chapter.

**Table 2-6** Shape-related functions described elsewhere

---

Book and chapter	Shape functions described
<i>Inside Macintosh: QuickDraw GX Objects</i> [this book]	
“Ink Objects”	GXGetShapeColor GXSetShapeColor GXGetShapeInkAttributes GXSetShapeInkAttributes GXGetShapeTransfer GXSetShapeTransfer
“Transform Objects”	GXGetShapeClip GXSetShapeClip GXGetShapeHitTest GXSetShapeHitTest GXGetShapeMapping GXSetShapeMapping GXGetShapeViewPorts GXSetShapeViewPorts GXMapShape GXMoveShape GXMoveShapeTo GXRotateShape GXScaleShape GXSkewShape

**Table 2-6** Shape-related functions described elsewhere (continued)

Book and chapter	Shape functions described
"View-Related Objects"	GXGetShapeDeviceArea
	GXGetShapeDeviceBounds
	GXGetShapeDeviceColors
	GXGetShapeGlobalBounds
	GXGetShapeGlobalViewPorts
	GXGetShapeGlobalViewDevices
	GXGetShapeLocalBounds
<i>Inside Macintosh: QuickDraw GX Graphics</i>	
"Geometric Shapes"	GXCountShapeContours
	GXCountShapePoints
	GXGetShapeFill
	GXSetShapeFill
	GXGetShapeIndex
	GXGetShapeParts
	GXSetShapeParts
	GXGetShapePoints
	GXSetShapePoints
	GXNewShapeVector
	GXSetShapeVector
"Geometric Styles"	GXGetShapeCap
	GXSetShapeCap
	GXGetShapeCurveError
	GXSetShapeCurveError
	GXGetShapeDash
	GXSetShapeDash
	GXGetShapeDashPositions
	GXGetShapeJoin
	GXSetShapeJoin
	GXGetShapePattern
	GXSetShapePattern
	GXGetShapePatternPositions
	GXGetShapePen
	GXSetShapePen
	GXGetShapeStyleAttributes
	GXSetShapeStyleAttributes

continued

## Shape Objects

**Table 2-6** Shape-related functions described elsewhere (continued)

Book and chapter	Shape functions described
"Geometric Operations"	GXContainsBoundsShape
	GXContainsShape
	GXDifferenceShape
	GXExcludeShape
	GXGetShapeArea
	GXGetShapeBounds
	GXSetShapeBounds
	GXGetShapeCenter
	GXGetShapeDirection
	GXGetShapeLength
	GXInsetShape
	GXIntersectShape
	GXInvertShape
	GXReduceShape
	GXReverseDifferenceShape
	GXReverseShape
	GXShapeLengthToPoint
	GXSimplifyShape
	GXTouchesBoundsShape
	GXTouchesShape
	GXUnionShape
"Bitmap Shapes"	GXGetShapePixel
	GXSetShapePixel
<i>Inside Macintosh: QuickDraw GX Typography</i>	
"Typographic Styles"	GXGetShapeDeviceFontMetrics
	GXGetShapeEncoding
	GXSetShapeEncoding
	GXGetShapeFace
	GXSetShapeFace
	GXGetShapeFont
	GXSetShapeFont
	GXGetShapeTextSize
	GXSetShapeTextSize
	GXGetShapeJustification
	GXSetShapeJustification
	GXGetShapeFontMetrics
	GXGetShapeFontVariations
	GXSetShapeFontVariations
	GXGetShapeFontVariationSuite
	GXGetShapeLocalFontMetrics
	GXGetShapeTextAttributes
	GXSetShapeTextAttributes
	GXGetShapeTypographicBounds



**Table 2-6** Shape-related functions described elsewhere (continued)

Book and chapter	Shape functions described
"Layout Shapes"	GXGetLayoutShapeParts GXSetLayoutShapeParts
"Layout Styles"	GXGetShapeRunControls GXSetShapeRunControls GXGetShapeRunFeatures GXSetShapeRunFeatures GXGetShapeRunGlyphSubstitutions GXSetShapeRunGlyphSubstitutions GXGetShapeRunKerningAdjustments GXSetShapeRunKerningAdjustments
"Layout Line Control"	GXGetShapeRunGlyphJustOverrides GXSetShapeRunGlyphJustOverrides GXGetShapeRunPriorityJustOverride GXSetShapeRunPriorityJustOverride
<i>Inside Macintosh: QuickDraw GX Environment and Utilities</i>	
"QuickDraw GX Debugging"	GXGetShapeDrawError GXValidateShape

## Shape Objects Reference

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

- type definitions of the data types, including enumerations, that are specific to shape objects
- descriptions of the QuickDraw GX functions that operate on shape objects in general, independent of the type of shape involved
- a description of an application-defined function used for flattening and unflattening shapes

## Constants and Data Types

This section describes the constants and the data types that you use to obtain and provide information about shape objects.

## Shape Objects

## The Shape Object

---

QuickDraw GX provides you with access to an individual shape object through a `gxShape` reference:

```
typedef struct gxPrivateShapeRecord *gxShape;
```

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

## Shape Type

---

A shape object's shape type specifies what type of geometry the shape object has. Constants for all shape types are defined in the `gxShapeTypes` enumeration:

```
enum gxShapeTypes {
    gxEmptyType = 1,
    gxPointType,
    gxLineType,
    gxCurveType,
    gxRectangleType,
    gxPolygonType,
    gxPathType,
    gxBitmapType,
    gxTextType,
    gxGlyphType,
    gxLayoutType,
    gxFullType,
    gxPictureType
};

typedef long gxShapeType;
```

The individual shape types are described further in Table 2-1 on page 2-9.

## Shape Fill

---

Each shape object has a shape fill property. The shape fill specifies how QuickDraw GX interprets the geometry of the shape: how the shape is drawn, how the shape is hit-tested, and how certain geometric operations, like the intersection operation, interpret the shape.

## Shape Objects

Constants for all shape fills are defined in the `gxShapeFills` enumeration:

```
enum gxShapeFills {
    gxNoFill,                /* shape not drawn */
    gxOpenFrameFill,         /* framed, one edge left open */
    gxFrameFill              = gxOpenFrameFill,
    gxClosedFrameFill,       /* framed, closed completely */
    gxHollowFill             = gxClosedFrameFill,
    gxEvenOddFill,           /* filled using even-odd rule */
    gxSolidFill              = gxEvenOddFill,
    gxWindingFill,           /* filled using winding-number rule */
    gxInverseEvenOddFill,    /* filled inverse of even-odd rule */
    gxInverseSolidFill       = gxInverseEvenOddFill,
    gxInverseFill            = gxInverseEvenOddFill,
    gxInverseWindingFill     /* filled inverse of winding-number */
};

typedef long gxShapeFill;
```

The individual shape fills are described further in Table 2-2 on page 2-13.

## Shape Attributes

Each shape object has a set of attributes. Shape *attributes* are a group of flags that modify the behavior of the shape object. Constants for all shape attributes are defined in the `gxShapeAttributes` enumeration:

```
enum gxShapeAttributes {
    gxNoAttributes,          /* no attributes set */
    gxDirectShape            = 0x0001, /* prefer GX heap */
    gxRemoteShape            = 0x0002, /* prefer accel. memory */
    gxCachedShape            = 0x0004, /* optimize drawing */
    gxLockedShape            = 0x0008, /* lock shape geometry */
    gxGroupShape             = 0x0010, /* treat as single shape */
    gxMapTransformShape      = 0x0020, /* alter transform */
    gxUniqueItemsShape       = 0x0040, /* copy picture items */
    gxIgnorePlatformShape    = 0x0080, /* use glyph codes */
    gxNoMetricsGridShape     = 0x0100, /* don't use hinting */
    gxDiskShape              = 0x0200, /* unload this first */
    gxMemoryShape            = 0x0400  /* unload this last */
};

typedef long gxShapeAttribute;
```

The individual shape attributes are described further in Table 2-4 on page 2-16.

## Flatten Flags

---

The flatten flags are used in a parameter to the `GXFlattenShape` function, to control the amount of font and bitmap information to include in a flattened shape. The flatten flags are defined in the `gxFlattenFlags` enumeration:

```
enum gxFlattenFlags {
    gxFontListFlatten      = 0x01,
    gxFontGlyphsFlatten   = 0x02,
    gxFontVariationsFlatten = 0x04,
    gxBitmapAliasFlatten   = 0x08
};
```

```
typedef long gxFlattenFlag;
```

### Constant descriptions

`gxFontListFlatten`

Instructs the `GXFlattenShape` function to attach to the flattened shape a tag object containing a list of the fonts referenced in the shape.

`gxFontGlyphsFlatten`

Instructs the `GXFlattenShape` function to attach to the flattened shape a tag object containing a list of the specific glyphs used from each font referenced by the shape.

`gxFontVariationsFlatten`

Instructs the `GXFlattenShape` function to attach to the flattened shape a tag object containing variation-axis coordinates describing all font variations used by the flattened shape.

`gxBitmapAliasFlatten`

Instructs the `GXFlattenShape` function to include with the flattened shape all image data from any bitmap shapes that are referenced by the shape. If this flag is not set, image data from bitmap shapes whose image data is disk-based is not included in the flattened shape, although the image data is not lost because a tag object specifying the file holding the image data is flattened along with the shape.

For more information on flattening shapes, see “Flattening and Unflattening Shapes” beginning on page 2-39. The `GXFlattenShape` function is described on page 2-88.

For information on font variations, see the font objects chapter of *Inside Macintosh: QuickDraw GX Typography*. For information on bitmap image data, see the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

## The Spool Block

The spool block structure is set up by an application before calling `GXFlattenShape` or `GXUnflattenShape`. Both the application and QuickDraw GX use and place values into the spool block.

```
struct gxSpoolBlock {
    gxSpoolProcPtr    spoolProcedure;
    void              *buffer;
    long              bufferSize;
    long              count;
    long              operationSize;
    long              operationOffset;
    gxGraphicsOpcode  lastTypeOpcode;
    gxGraphicsOpcode  currentOperation;
    gxGraphicsOpcode  currentOperand;
    unsigned char      compressed;
};
```

### Field descriptions

**spoolProcedure** A pointer to an application-defined function that either saves the flattened data or supplies the data for unflattening. The `gxSpoolProcPtr` type is defined as follows:

```
typedef long (*gxSpoolProcPtr)
              (gxSpoolCommand command,
               struct gxSpoolBlock *block);
```

	The format for the spool function is described on page 2-91.
<b>buffer</b>	A pointer to a buffer that holds the flattened data, after flattening or before unflattening. In either case the buffer is allocated by the application.
<b>bufferSize</b>	The size of the buffer. (Set by the application.)
<b>count</b>	The number of bytes of data read into or out of the buffer. (Set by QuickDraw GX.)
<b>operationSize</b>	The size of the current operation in the flattened stream. It is equal to the size field of the operand of the current operation. For flattening, it is the amount of data that QuickDraw GX will place into the buffer to complete the current operation; for unflattening, it is the amount of information that the spool function must place in the buffer to complete the current operation. (Set by QuickDraw GX.)

## Shape Objects

## operationOffset

For flattening, the offset in bytes from the beginning of the current operation to the end of the data currently in the buffer. For unflattening, the offset in bytes from the beginning of the current operation to the start of the data that needs to be placed in the buffer. It is the amount of the current operation that has so far been flattened or is about to be unflattened. (Set by QuickDraw GX.)

## lastTypeOpcode

The type of object currently being flattened or unflattened. It is one of the constants defined in the `gxGraphicsNewOpcode` enumeration. (Set by QuickDraw GX.)

## currentOperation

The type of operation currently being flattened or unflattened. It is one of the constants defined in the `gxGraphicsOperationOpcode` enumeration. (Set by QuickDraw GX.)

## currentOperand

The type of data (within the current object) being flattened or unflattened. It is one of the constants defined in one of the data opcode enumerations, such as the `gxShapeDataOpcode` enumeration or the `gxStyleDataOpcode` enumeration. (Set by QuickDraw GX.)

## compressed

The type of compression applied to the current item. (Set by QuickDraw GX.)

General information about flattening shapes is found in the section “Flattening and Unflattening Shapes” beginning on page 2-39. The `GXFlattenShape` function is described on page 2-88. The `GXUnflattenShape` function is described on page 2-90. The QuickDraw GX stream format, including the opcodes it uses and the types of compression it supports, is described in the stream format chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## The Hit-Test Info Structure

---

The hit-test info structure is a structure in which both the `GXHitTestShape` and `GXHitTestPicture` functions return information. `GXHitTestShape` uses only the first three fields; `GXHitTestPicture` uses all seven fields.

```
struct gxHitTestInfo {
    gxShapePart    what;
    long           index;
    Fixed          distance;
    gxShape        which;
    gxShape        containerPicture;
    long           containerIndex;
    long           totalIndex;
};
```

## Shape Objects

**Field descriptions**

<code>what</code>	The parts of the shape that were hit, if any. QuickDraw GX returns in this field a mask denoting all shape parts (out of those specified for the hit-test) that are within the tolerance of the hit-test from the hit point. Shape parts are defined in the <code>gxShapeParts</code> enumeration; the tolerance and the subset of shape parts to test for make up the hit-test parameters. All are described in the chapter “Transform Objects” in this book.
<code>index</code>	The index of the nearest point in the geometry to the hit point. Every point in a shape’s geometry has an index number (indexes start at 1).
<code>distance</code>	The distance in geometry units from the hit point to the closest point on the shape part that was hit. (If no part was hit, this value is undefined.) If more than one shape part was hit, this is the distance to the first shape part encountered that is within the tolerance of the hit point. The order in which shape parts are examined during hit-testing is defined by the <code>gxShapeParts</code> enumeration, described in the chapter “Transform Objects” in this book.
<code>which</code>	A reference to the specific shape that was hit. (Used only by <code>GXHitTestPicture</code> .)
<code>containerPicture</code>	A reference to the picture shape that immediately contains the specific shape that was hit. Note that this may be a picture shape contained at some level within the picture shape specified in the call to <code>GXHitTestPicture</code> . (Used only by <code>GXHitTestPicture</code> .)
<code>containerIndex</code>	The index number—within the immediately containing shape—of the specific shape that was hit. (Used only by <code>GXHitTestPicture</code> .)
<code>totalIndex</code>	The index number—within the picture shape specified in the call to <code>GXHitTestPicture</code> —of the specific shape that was hit. (Used only by <code>GXHitTestPicture</code> .)

The `GXHitTestShape` function is described on page 2-86. The `GXHitTestPicture` function is described in the picture shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

## Functions

---

This section describes the QuickDraw GX functions you can use to

- create and manipulate a shape object
- manipulate the properties of a shape object, including converting a shape from one type to another
- directly manipulate a shape’s geometry
- flatten and unflatten a shape
- draw and hit-test a shape

## Shape Objects

**Note**

Shape-related QuickDraw GX functions not described in this section are listed and cross-referenced in Table 2-6 on page 2-42. ♦

## Creating and Manipulating Shape Objects

---

The functions described in this section allow you to work with shapes as objects in memory. With the functions in this section, you can

- determine the default shape object
- create and dispose of a shape object
- find the size of a shape object in memory
- copy, clone, and compare shape objects
- cache a shape object

### *GXGetDefaultShape*

---

You can use the `GXGetDefaultShape` function to obtain a reference to the default shape object for a particular shape type.

```
gxShape GXGetDefaultShape(gxShapeType aType);
```

`aType`            A shape type that specifies which default shape object to return.

*function result*   A reference to the default shape for the shape type specified by the `aType` parameter.

**DESCRIPTION**

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

You can also alter a default shape object by using the `GXSetDefaultShape` function.

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

`out_of_memory`  
`illegal_type_for_shape`        (debugging version)



**SEE ALSO**

Default shape objects are discussed in the section “Default Shapes” beginning on page 2-18.

The `GXSetDefaultShape` function is described in the next section.

To create a copy of a default shape object, use the `GXNewShape` function, described on page 2-54.

***GXSetDefaultShape***

---

You can use the `GXSetDefaultShape` function to replace the default shape object of a particular shape type.

```
void GXSetDefaultShape(gxShape target);
```

`target`      A reference to the new default shape object.

**DESCRIPTION**

The `GXSetDefaultShape` function replaces an existing default shape with the shape specified by the `target` parameter. The shape type of the target shape determines which default shape is replaced. This function disposes of the old default shape and increments the owner count of the target shape.

You can use the `GXSetDefaultShape` function to replace the style, ink, or transform of one of the default shapes by specifying a target shape with a different style, ink, or transform than the old default shape. When QuickDraw GX creates new shapes of the target shape’s shape type, the new shape will have the same ink, style, and transform as the target shape.

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

```
out_of_memory
shape_is_nil
illegal_type_for_shape      (debugging version)
```

**SEE ALSO**

Default shape objects are discussed in the section “Default Shapes” beginning on page 2-18.

To create a copy of a default shape object, use the `GXNewShape` function, described in the next section.

***GXNewShape***

---

You can use the `GXNewShape` function to create a new shape of a specified shape type.

```
gxShape GXNewShape (gxShapeType aType) ;
```

`aType`            The type of shape object to create.

*function result*   A reference to a newly created copy of the default shape object of the type specified by the `aType` parameter.

***DESCRIPTION***

The `GXNewShape` function creates a copy of the default shape object of the type specified by the `aType` parameter and gives it an owner count of 1.

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, ink, and transform of the shape by using the functions `GXSetShapeStyle`, `GXSetShapeInk`, and `GXSetShapeTransform`.

You can use this function by itself to create empty and full shapes. For other shape types, you can use this function to create a shape and then you can customize the shape's geometry by using additional functions, such as `GXSetShapeGeometry` or one of the shape-specific functions such as `GXSetPoint`, `GXSetLine`, `GXSetPathParts`, or `GXSetGlyphParts`.

***SPECIAL CONSIDERATIONS***

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

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

`out_of_memory`  
`illegal_type_for_shape`      (debugging version)

***SEE ALSO***

Shape types, including empty and full shapes, are described in the section "Shape Type" beginning on page 2-9.

Default shape objects are discussed in the section "Default Shapes" beginning on page 2-18. To examine a default shape, use the `GXGetDefaultShape` function, described on page 2-52. To replace a default shape, use the `GXSetDefaultShape` function, described on page 2-53.

## Shape Objects

The `GXSetShapeStyle` function is described on page 2-70; the `GXSetShapeInk` function is described on page 2-71; the `GXSetShapeTransform` function is described on page 2-73.

The `GXSetShapeGeometry` function is described on page 2-67. Other geometry-setting functions are described in the shape-specific chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*.

For an example of the use of this function, see page 2-24.

## ***GXDisposeShape***

---

You can use the `GXDisposeShape` function to release a reference to a shape.

```
void GXDisposeShape(gxShape target);
```

`target`      A reference to the shape to dispose of.

### **DESCRIPTION**

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

### **SPECIAL CONSIDERATIONS**

You cannot dispose of a shape that is locked, either because the `gxLockedShape` attribute is set or because `GXLockShape` was called to lock the shape. Depending on how the shape became locked, you must call `GXSetShapeAttributes` or `GXUnlockShape` before calling `GXDisposeShape` on a locked shape.

### **ERRORS, WARNINGS, AND NOTICES**

#### **Errors**

`shape_is_nil`  
`shape_access_not_allowed`      (debugging version)

#### **Warnings**

`cannot_dispose_default_shape`      (debugging version)

## Shape Objects

**SEE ALSO**

Owner counts are discussed in the section “Copying, Comparing, and Cloning Shape Objects” beginning on page 2-25, and in the section “Manipulating a Shape Object’s Owner Count” beginning on page 2-31.

To examine the owner count of a shape, use the `GXGetShapeOwners` function, described on page 2-76. To increment the owner count of a shape, use the `GXCloneShape` function, described on page 2-61.

For an example of the use of this function, see page 2-24.

***GXGetShapeSize***

---

You can use the `GXGetShapeSize` function to determine the amount of memory currently occupied by a shape object.

```
long GXGetShapeSize(gxShape source);
```

*source*            A reference to the shape object to determine the current memory size of.

*function result*   The number of bytes of memory currently occupied by the shape specified in the *source* parameter.

**DESCRIPTION**

The `GXGetShapeSize` function takes the source shape’s type, owner count, fill, attributes, and geometry into consideration. It does not include the memory used by the shape’s style, ink, transform, or tag objects, but does include the memory used by the references to them.

The function result also includes the size of some shape properties private to QuickDraw GX, but does not include the size of the shape cache or the size of any memory overhead used to represent the shape.

This function returns only the memory size currently used by the shape. For example, when a shape is unloaded to disk it uses less memory, and the result of this function reflects its smaller size.

You can use the `GXLoadShape` function to load a shape into memory before determining its size.

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

`shape_is_nil`

**SEE ALSO**

To find the size of a shape's cache, use the `GXGetShapeCacheSize` function, described on page 2-64.

The `GXLoadShape` function is described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

For information about the memory size of graphic shapes, see the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. For information about the memory size of typographic shapes, see the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*.

***GXCopyToShape***

---

The `GXCopyToShape` function copies the contents of one existing shape to another or else it creates a new shape and copies the contents of an existing shape to it. You can, for example, use this function to create a copy of a shape so that you can modify it without changing the original shape.

```
gxShape GXCopyToShape(gxShape target, gxShape source);
```

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

**source**        A reference to the shape to copy from.

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

**DESCRIPTION**

The `GXCopyToShape` function copies the properties and the geometry of the shape specified by the `source` parameter into the shape specified by the `target` parameter. It also copies the references to the source shape's ink, style, transform, and tags; that is, after the function returns, the target shape and the source shape share the same ink, style, transform, and tag objects. This function increments by 1 the owner counts of the source shape's ink, style, transform, and tag objects, and disposes of the original ink, style, transform, and tags of the target shape.

If you specify `nil` for the `target` parameter, this function creates a new shape to copy the contents of the source shape into.

**SPECIAL CONSIDERATIONS**

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

## Shape Objects

If the target shape is locked, the `GXCopyToShape` function posts a `shape_access_not_allowed` error. If you try to copy a picture into a shape that is contained in the picture, this function posts a `picture_cannot_contain_itself` error. If you try to copy a shape of one type into the default shape of another type, this function posts a `cannot_dispose_default_shape` warning.

This function does not copy the pixel image of bitmap shapes, the shapes contained within picture shapes, or the set of style objects associated with glyph or layout shapes; instead, it copies the references to them. To obtain a complete copy of a bitmap shape, picture shape, glyph shape, or layout shape, use the `GXCopyDeepToShape` function.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

`out_of_memory`

`shape_is_nil`

`shape_access_not_allowed` (debugging version)

`picture_cannot_contain_itself` (debugging version)

**Warnings**

`cannot_dispose_default_shape` (debugging version)

## SEE ALSO

To create a new shape that is a copy of the default shape instead of a copy of an existing shape, use the `GXNewShape` function, described on page 2-54.

The `GXCopyDeepToShape` function copies the pixel image of bitmap shapes and the shapes contained within picture shape; it is described in the next section. For more information about copying bitmap shapes and picture shapes, see *Inside Macintosh: QuickDraw GX Graphics*.

For information about copying typographic shapes, see the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*.

***GXCopyDeepToShape***

---

The `GXCopyDeepToShape` function copies the contents of one existing shape to another, or creates a new shape and copies the contents of an existing shape to it. For bitmap shapes, picture shapes, glyph shapes, and layout shapes, `GXCopyDeepToShape` copies more information than the `GXCopyToShape` function does.

```
gxShape GXCopyDeepToShape(gxShape target, gxShape source);
```

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

**source**        A reference to the shape to copy from.

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

**DESCRIPTION**

The `GXCopyDeepToShape` function copies the properties and the geometry of the shape specified by the `source` parameter into the shape specified by the `target` parameter. It also copies the references to the source shape's ink, style, transform, and tags; that is, after the function returns, the target shape and the source shape share the same ink, style, and transform objects. This function increments by 1 the owner counts of the source shape's ink, style, and transform, and disposes of the ink, style, and transform of the target shape.

If you specify `nil` for the `target` parameter, this function creates a new shape to copy the contents of the source shape into.

The `GXCopyDeepToShape` function is similar to the `GXCopyToShape` function except that it performs these additional operations:

- For bitmap shapes, `GXCopyDeepToShape` also copies the complete pixel image.
- For picture shapes, `GXCopyDeepToShape` also copies each shape in the source picture. If the source picture contains other picture shapes, their shapes are also recursively copied. The styles, inks, and transforms of the shapes within the picture are not copied; instead the copied shapes share references to the styles, inks, and transforms of the original shapes, and the `GXCopyDeepToShape` function increments by 1 the owner counts of the original styles, inks, and transforms.
- For glyph and layout shapes, `GXCopyDeepToShape` also copies the set of style objects referenced in the style list that is part of the shape's geometry.

Because the `GXCopyDeepToShape` function copies the pixel image of bitmap shapes and the shapes contained within picture shapes, you can use it to create a copy of a bitmap or a picture, and then modify the copy without changing the original bitmap or picture.

**SPECIAL CONSIDERATIONS**

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

If you try to copy a picture into a shape that is contained in the picture, the `GXCopyDeepToShape` function posts a `picture_cannot_contain_itself` error. If the target shape is locked, this function posts a `shape_access_not_allowed` error. If you try to copy a shape of one type into the default shape of another type, this function posts a `cannot_dispose_default_shape` warning.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

`out_of_memory`  
`shape_is_nil`  
`shape_access_not_allowed` (debugging version)  
`picture_cannot_contain_itself` (debugging version)

**Warnings**

`cannot_dispose_default_shape` (debugging version)

## SEE ALSO

To create a new shape that is a copy of the default shape instead of a copy of an existing shape, use the `GXNewShape` function, described on page 2-54.

To make a copy of an existing shape without copying all information for bitmap shapes, picture shapes, glyph shape, and layout shapes, use the `GXCopyToShape` function, described in the previous section.

For information about copying typographic shapes, see the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*.

***GXEqualShape***

---

You can use the `GXEqualShape` function to determine if two shapes are equal.

```
boolean GXEqualShape(gxShape one, gxShape two);
```

`one`            A reference to one of the shapes to test for equality.

`two`            A reference to the other shape to test for equality.

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

## DESCRIPTION

The `GXEqualShape` function returns as its function result a Boolean value indicating whether the two QuickDraw GX shapes are equal. For two QuickDraw GX shapes to be equal, they must satisfy these requirements:

- They must have the same shape type and fill, but they do not need to have the same attributes, owner count, or tag list.
- Their geometries must have identical values; geometries that are equivalent but not identical are not considered to be equal. To eliminate false rejection of equivalent geometries, call the `GXSimplifyShape` function to simplify both shapes before you call `GXEqualShape`.



## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory  
shape\_is\_nil

## SEE ALSO

Equivalent geometries are described in the section “Copying, Comparing, and Cloning Shape Objects” beginning on page 2-25. The `GXSimplifyShape` function is described in the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

To make a copy of a shape object that is equal by the criteria of this function, use the `GXCopyToShape` function, described on page 2-57.

***GXCloneShape***

---

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

```
gxShape GXCloneShape(gxShape source);
```

`source`        A reference to the shape to clone.

*function result* A reference to the cloned shape.

**DESCRIPTION**

The `GXCloneShape` function returns a reference to the shape object specified by the `source` parameter and increments its owner count by 1. You typically use this function when you want to create another reference to an existing shape rather than create a distinct copy of the shape.

This function returns as its function result a reference to the shape—the same reference you pass in as the `source` parameter. Thus you can clone a shape with the following line of C code:

```
aShapeClone = GXCloneShape(aShape);
```

This line of code has almost the same affect as

```
aShapeClone = aShape;
```

that is, it sets the `aShapeClone` variable to reference the same shape object as the `aShape` variable. The difference is that `GXCloneShape` also increments the shape’s owner count.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**`shape_is_nil`

## SEE ALSO

Owner counts are discussed in the section “Copying, Comparing, and Cloning Shape Objects” beginning on page 2-25, and in the section “Manipulating a Shape Object’s Owner Count” beginning on page 2-31.

To examine the owner count of a shape, use the `GXGetShapeOwners` function, described on page 2-76. To decrement the owner count of a shape, use the `GXDisposeShape` function, described on page 2-55.

***GXCacheShape***

---

You can use the `GXCacheShape` function to prepare a shape for faster drawing.

```
void GXCacheShape(gxShape source);
```

`source`      A reference to the shape to build the cache for.

**DESCRIPTION**

The `GXCacheShape` function prepares a shape for drawing by performing the calculations necessary to draw the shape and storing them in a shape cache. Then, when you draw the shape, time is saved because those calculations have already been made.

Although you do not need to call this function before drawing, you can use it to improve the speed of drawing on the screen.

To build a shape cache, use this function. To delete a shape cache, use the `GXDisposeShapeCache` function. To determine the amount of memory occupied by a shape cache, use the `GXGetShapeCacheSize` function.

**SPECIAL CONSIDERATIONS**

If you set the `gxCachedShape` attribute for a shape, QuickDraw GX automatically creates a cache and a compressed offscreen bitmap for the shape the first time it draws the shape. Unlike calling `GXCacheShape`, setting the `gxCachedShape` attribute can result in increased memory requirements for a shape.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

Because it performs preliminary calculations involved in drawing, the `GXCacheShape` function can, in addition to the errors listed below, post any errors and warnings associated with the `GXDrawShape` function. Therefore, `GXCacheShape` can post font-related errors if it is caching text.

**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_bitmap_exceeds_implementation_limit</code>	
<code>pattern_lattice_out_of_range</code>	(debugging version)

**Warnings**

<code>character_substitution_took_place</code>	
<code>graphic_type_cannot_be_dashed</code>	(debugging version)
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)
<code>unable_to_draw_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)
<code>face_override_style_font_must_match_style</code>	(debugging version)

## SEE ALSO

Shape caches are discussed in the section “Caching Shape Objects” beginning on page 2-27. The `gxCachedShape` attribute is described in that section and also in Table 2-4 on page 2-16.

The `GXGetShapeCacheSize` function is described on page 2-64. The `GXDisposeShapeCache` function is described in the next section.

For information about the caching and drawing typographic shapes, see the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*.

***GXDisposeShapeCache***

You can use the `GXDisposeShapeCache` function to release the memory occupied by a shape’s cache.

```
void GXDisposeShapeCache(gxShape target);
```

`target`      A reference to the shape whose cache is to be disposed of.

## Shape Objects

**DESCRIPTION**

The `GXDisposeShapeCache` function immediately releases the memory allocated to the cache of the shape indicated by the `target` parameter. This function releases only that memory allocated to the target shape's cache. It does not release memory allocated to any related system caches or globals.

To build a shape cache, use the `GXCacheShape` function. To delete a shape cache, use this function. To determine the amount of memory occupied by a shape cache, use the `GXGetShapeCacheSize` function.

**SPECIAL CONSIDERATIONS**

You never need to call this function. QuickDraw GX disposes of caches automatically when it needs additional memory.

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

`shape_is_nil`

**SEE ALSO**

Shape caches are discussed in the section “Caching Shape Objects” beginning on page 2-27. The `GXCacheShape` function is described on page 2-62. The `GXGetShapeCacheSize` function is described in the next section.

***GXGetShapeCacheSize***

---

You can use the `GXGetShapeCacheSize` function to determine how much memory is allocated to a shape and its cache.

```
long GXGetShapeCacheSize(gxShape source);
```

**source**      A reference to the shape object whose size in memory (including cache) is to be determined.

*function result*    The approximate number of bytes of memory currently occupied by the shape and cache referenced in the `source` parameter.

**DESCRIPTION**

The `GXGetShapeCacheSize` function, like the `GXGetShapeSize` function, calculates the size of the source shape in memory, and does not include the memory used by the shape's referenced tags, style, ink, or transform. However, unlike `GXGetShapeSize`, this function result also includes the size of the source shape's current cache.

## Shape Objects

This function returns only the memory size currently being used by the shape and its cache. If the shape is unloaded to disk, the result of this function indicates the smaller amount of memory used. If the shape has no cache, the result of this function is simply the memory size of the shape. You can use the `GXLoadShape` function to load a shape into memory before calling this function, to get the full size of the shape and cache.

In the interest of speed, this function provides only an approximation of the memory requirements of the shape's cache. The actual memory requirements of the cache depend on many factors, such as memory overhead, and would be less efficient to calculate. You can use this function to determine an approximate size for the memory partition needed for a set of shapes to be loaded and cached at the same time.

To determine the amount of memory occupied by a shape and its cache, use this function. To build a shape cache, use the `GXCacheShape` function. To delete a shape cache, use the `GXDisposeShapeCache` function.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

`shape_is_nil`

## SEE ALSO

Shape caches are discussed in the section “Caching Shape Objects” beginning on page 2-27. The `GXCacheShape` function is described on page 2-62. The `GXDisposeShapeCache` function is described in the previous section.

The `GXLoadShape` function is described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To determine the amount of memory occupied by a shape without its cache, use the `GXGetShapeSize` function, described on page 2-56.

## Manipulating Shape Object Properties

---

This section describes the functions available for manipulating the properties of shape objects. The functions described in this section allow you to

- determine a shape object's type, geometry, and fill
- determine the style, ink, and transform objects associated with a shape
- determine a shape object's attributes
- reset certain shape properties to their default values
- find the owner count of a shape object
- determine the tag objects associated with a shape object

Functions for direct manipulation of the geometry property of a shape object are described in the next section, “Directly Manipulating a Shape's Geometry” beginning on page 2-80.

## *GXGetShapeType*

---

You can use the `GXGetShapeType` function to determine the shape type of a shape object.

```
gxShapeType GXGetShapeType(gxShape source);
```

`source`        A reference to the shape object to determine the shape type of.

*function result*   The shape type of the source shape.

### *ERRORS, WARNINGS, AND NOTICES*

#### **Errors**

`shape_is_nil`

### *SEE ALSO*

Shape types are described in the section “Shape Type” beginning on page 2-9.

To assign a shape type to a shape object, use the `GXSetShapeType` function, described in the next section.

## *GXSetShapeType*

---

You can use the `GXSetShapeType` function to convert a shape object from one shape type to another.

```
void GXSetShapeType(gxShape target, gxShapeType newType);
```

`target`        A reference to the shape object to assign the new shape type to.

`newType`       A reference to the shape type to be assigned to the shape.

### *DESCRIPTION*

The `GXSetShapeType` function changes the type of the target shape to the shape type specified by `newType`. Many different kinds of conversions are possible: typographic types can be converted to other typographic types or to graphic types; graphic types can be converted to other graphic types. The results of the conversion differ in each case, depending on which type is converted to which other type. See Table 2-5 on page 2-33 for a list of chapters that describe how conversion works for different shape types.

**ERRORS, WARNINGS, AND NOTICES**

When you change a shape to a bitmap type, the `GXSetShapeType` function performs preliminary calculations on its data and thus may post, in addition to the errors listed below, errors associated with the `GXDrawShape` function. When you change a shape to a typographic type, `GXSetShapeType` may post font-related errors.

**Errors**

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

**Warnings**

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

**Notices (debugging version)**

<code>shape_type_already_set</code>	
-------------------------------------	--

**SEE ALSO**

What happens when you call `GXSetShapeType` to convert shapes of one type to another is described in *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*. Table 2-5 on page 2-33 shows which specific chapters to read for detailed information on conversion among the various types of shapes.

Shape types are described in the section “Shape Type” beginning on page 2-9 of this chapter.

To determine the shape type of a shape object, use the `GXGetShapeType` function, described in the previous section.

***GXSetShapeGeometry***

---

You can use the `GXSetShapeGeometry` function to copy the geometry from one shape object to another.

```
void GXSetShapeGeometry(gxShape target, gxShape geometry);
```

<code>target</code>	A reference to the shape to copy the new geometry into.
<code>geometry</code>	A reference to the shape to copy the new geometry from.

**DESCRIPTION**

For two shape objects with the same shape type, the `GXSetShapeGeometry` function copies the geometry from the shape referenced by the `geometry` parameter to the shape referenced by the `target` parameter. If the type of the shape referenced in the `geometry` parameter is different from the type of the target shape, the target shape becomes the geometry shape’s type.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory  
 shape\_is\_nil  
 picture\_cannot\_contain\_itself  
 shape\_access\_not\_allowed (debugging version)

## SEE ALSO

To directly manipulate the contents of a shape’s geometry, see the section “Directly Manipulating a Shape’s Geometry” beginning on page 2-34; see also the descriptions of the `GXLockShape`, `GXUnlockShape`, `GXGetShapeStructure`, and `GXChangedShape` functions, beginning on page 2-80.

Specific methods for setting and manipulating the geometries of graphic shapes are described in *Inside Macintosh: QuickDraw GX Graphics*. Methods for setting and manipulating the geometries of typographic shapes are described in *Inside Macintosh: QuickDraw GX Typography*.

***GXGetShapeFill***

---

You can use the `GXGetShapeFill` function to retrieve the fill property of a shape object.

```
gxShapeFill GXGetShapeFill(gxShape source);
```

`source`      A reference to the shape whose fill property you want to retrieve.

*function result*   The fill of the source shape.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory  
 shape\_is\_nil

## SEE ALSO

Shape fills are described in the section “Shape Fill” beginning on page 2-13.

To assign a fill to a shape object, use the `GXSetShapeFill` function, described in the next section.

For more information on shape fill as it applies to geometric shapes, see the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. For more information on shape fill as it applies to typographic shapes, see the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*.



## *GXSetShapeFill*

---

You can use the `GXSetShapeFill` function to change the fill property of a shape object.

```
void GXSetShapeFill(gxShape target, gxShapeFill newFill);
```

`target`            A reference to the shape whose fill property you want to change.

`newFill`          The new value for shape fill.

### ERRORS, WARNINGS, AND NOTICES

#### Errors

`out_of_memory`

`shape_is_nil`

`shape_access_not_allowed`      (debugging version)

`parameter_out_of_range`        (debugging version)

`inconsistent_parameters`       (debugging version)

#### Notices (debugging version)

`fill_already_set`

### SEE ALSO

Shape fills are described in the section “Shape Fill” beginning on page 2-13.

This function is further described for geometric shapes in the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*, and for typographic shapes in the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*.

To determine the shape fill of a shape object, use the `GXGetShapeFill` function, described in the previous section.

## *GXGetShapeStyle*

---

You can use the `GXGetShapeStyle` function to determine the style object associated with a QuickDraw GX shape.

```
gxStyle GXGetShapeStyle(gxShape source);
```

`source`           A reference to the shape object whose style object is to be determined.

*function result* A reference to the style object associated with the source shape object.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory  
shape\_is\_nil

## SEE ALSO

The relationship of style objects to QuickDraw GX shapes is discussed in “About QuickDraw GX Shapes” beginning on page 2-5. Style objects themselves are further discussed in the chapter “Style Objects” in this book.

To change the style object associated with a QuickDraw GX shape, use the `GXSetShapeStyle` function, described in the next section.

***GXSetShapeStyle***

---

You can use the `GXSetShapeStyle` function to change the style object associated with a QuickDraw GX shape.

```
void GXSetShapeStyle(gxShape target, gxStyle newStyle);
```

`target`        A reference to the shape whose style object is to be changed.  
`newStyle`     A reference to the new style object to associate with the target shape.

## DESCRIPTION

The `GXSetShapeStyle` function disassociates the style object already associated with the target shape and disposes of it. The function then assigns the style object referenced by the `newStyle` parameter to the target shape and increments by 1 the owner count of the new style object.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory  
shape\_is\_nil  
style\_is\_nil  
shape\_access\_not\_allowed     (debugging version)

**Notices (debugging version)**

style\_already\_set

**SEE ALSO**

The relationship of style objects to QuickDraw GX shapes is discussed in “About QuickDraw GX Shapes” beginning on page 2-5. Style objects themselves are further discussed in the chapter “Style Objects” in this book.

To determine the style object associated with a QuickDraw GX shape, use the `GXGetShapeStyle` function, described in the previous section.

***GXGetShapeInk***

---

You can use the `GXGetShapeInk` function to determine the ink object associated with a shape object.

```
gxInk GXGetShapeInk(gxShape source);
```

`source`      A reference to the shape whose ink object is to be determined.

*function result*   A reference to the ink object associated with the source shape object.

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

`out_of_memory`  
`shape_is_nil`

**SEE ALSO**

The relationship of ink objects to QuickDraw GX shapes is discussed in “About QuickDraw GX Shapes” beginning on page 2-5. Ink objects themselves are further discussed in the chapter “Ink Objects” in this book.

To change the ink object associated with a QuickDraw GX shape, use the `GXSetShapeInk` function, described in the next section.

***GXSetShapeInk***

---

You can use the `GXSetShapeInk` function to change the ink object associated with a shape object.

```
void GXSetShapeInk(gxShape target, gxInk newInk);
```

`target`      A reference to the shape whose ink object is to be changed.

`newInk`      A reference to the new ink object to associate with the target shape.

## Shape Objects

**DESCRIPTION**

The `GXSetShapeInk` function disassociates the ink object already associated with the target shape and disposes of it. The function then assigns the ink object referenced by the `newInk` parameter to the target shape and increments the owner count of the new ink object.

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

`out_of_memory`  
`shape_is_nil`  
`ink_is_nil`  
`shape_access_not_allowed` (debugging version)

**Notices (debugging version)**

`ink_already_set`

**SEE ALSO**

The relationship of ink objects to QuickDraw GX shapes is discussed in “About QuickDraw GX Shapes” beginning on page 2-5. Ink objects themselves are further discussed in the chapter “Ink Objects” in this book.

To determine the ink object associated with a QuickDraw GX shape, use the `GXGetShapeInk` function, described in the previous section.

***GXGetShapeTransform***

---

You can use the `GXGetShapeTransform` function to determine the transform object associated with a shape object.

```
gxTransform GXGetShapeTransform(gxShape source);
```

*source*        A reference to the shape whose transform object is to be determined.

*function result* A reference to the transform object associated with the source shape object.

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

`out_of_memory`  
`shape_is_nil`

**SEE ALSO**

The relationship of transform objects to QuickDraw GX shapes is discussed in “About QuickDraw GX Shapes” beginning on page 2-5. Transform objects themselves are further discussed in the chapter “Transform Objects” in this book.

To change the transform object associated with a QuickDraw GX shape, use the `GXSetShapeTransform` function, described in the next section.

***GXSetShapeTransform***

You can use the `GXSetShapeTransform` function to change the transform object associated with a shape object.

```
void GXSetShapeTransform(gxShape target,
                        gxTransform newTransform);
```

`target`            A reference to the shape whose transform object is to be changed.

`newTransform`

A reference to the new transform object to associate with the target shape.

**DESCRIPTION**

The `GXSetShapeTransform` function disassociates the transform object already associated with the target shape and disposes of it. `GXSetShapeTransform` then assigns the transform object referenced by the `newTransform` parameter to the target shape and increments by 1 the owner count of the new transform object.

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

`out_of_memory`  
`shape_is_nil`  
`transform_is_nil`  
`shape_access_not_allowed`      (debugging version)

**Notices (debugging version)**

`transform_already_set`

**SEE ALSO**

The relationship of transform objects to QuickDraw GX shapes is discussed in “About QuickDraw GX Shapes” beginning on page 2-5. Transform objects themselves are further discussed in the chapter “Transform Objects” in this book.

To determine the transform object associated with a QuickDraw GX shape, use the `GXGetShapeTransform` function, described in the previous section.

## ***GXGetShapeAttributes***

---

You can use the `GXGetShapeAttributes` function to examine which attributes of a shape object are set.

```
gxShapeAttribute GXGetShapeAttributes(gxShape source);
```

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

*function result*   The shape attributes of the source shape.

### **ERRORS, WARNINGS, AND NOTICES**

#### **Errors**

`out_of_memory`  
`shape_is_nil`

### **SEE ALSO**

Shape attributes are described in the section “Shape Attributes” beginning on page 2-16, and in the section “Getting and Setting a Shape Object’s Type, Fill, and Attributes” beginning on page 2-28.

To change the attributes of a shape object, use the `GXSetShapeAttributes` function, described in the next section.

For an example of the use of this function, see page 2-29.

## ***GXSetShapeAttributes***

---

You can use the `GXSetShapeAttributes` function to set or clear the attributes for a particular shape object.

```
void GXSetShapeAttributes(gxShape target, gxShapeAttribute
attributes);
```

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

`attributes`   The new shape attributes to be assigned.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

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

**Warnings**

`picture_expected`  
`cannot_set_unique_items_attribute_when_picture_contains_items`

## SEE ALSO

Shape attributes are described in the section “Shape Attributes” beginning on page 2-16, and in the section “Getting and Setting a Shape Object’s Type, Fill, and Attributes” beginning on page 2-28.

To examine the attributes of a shape object, use the `GXGetShapeAttributes` function, described in the previous section.

For an example of the use of this function, see page 2-29.

***GXResetShape***

---

You can use the `GXResetShape` function to reset the attributes, fill, style, ink, and transform of a shape to their default values.

```
void GXResetShape(gxShape target);
```

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

**DESCRIPTION**

The `GXResetShape` function resets the shape attributes and the shape fill of the shape object specified by the `target` parameter to match the shape attributes and shape fill of the corresponding default shape. The function also resets the style, ink, and transform references of the target shape to their default values.

The `GXResetShape` function does not change the target shape’s geometry, owner count, or tags.

After the `GXResetShape` function returns, the target shape references the same style, ink, and transform as the corresponding default shape object. The `GXResetShape` function increments by 1 the owner counts of the default style, ink, and transform, and disposes of the target shape’s original style, ink, and transform.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

`out_of_memory`  
`shape_is_nil`  
`shape_access_not_allowed` (debugging version)

## SEE ALSO

Default shape objects are described in the section “Default Shapes” beginning on page 2-18.

To examine a default shape, use the `GXGetDefaultShape` function, described on page 2-52. To replace a default shape, use the `GXSetDefaultShape` function, described on page 2-53.

For information on resetting typographic shapes, see the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*.

***GXGetShapeOwners***

---

You can use the `GXGetShapeOwners` function to determine the number of references to a particular shape object.

```
long GXGetShapeOwners(gxShape source);
```

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

*function result*   The owner count of the source shape.

## DESCRIPTION

The `GXGetShapeOwners` function returns as its function result the current number of references to the shape object.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

`shape_is_nil`

## SEE ALSO

Owner counts for shape objects are discussed in the section “Copying, Comparing, and Cloning Shape Objects” beginning on page 2-25, and in the section “Manipulating a Shape Object’s Owner Count” beginning on page 2-31.



## Shape Objects

To increment the owner count of a shape, use the `GXCloneShape` function, described on page 2-61. To decrement the owner count of a shape, use the `GXDisposeShape` function, described on page 2-55.

## ***GXGetShapeTags***

---

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

```
long GXGetShapeTags(gxShape source, long tagType, long index,
                    long count, gxTag items[]);
```

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

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

### ***DESCRIPTION***

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

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

If you pass a value other than `nil` for the `items` parameter, the `GXGetShapeTags` function returns in it the tag references that were found.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

```

out_of_memory
shape_is_nil
index_is_less_than_one    (debugging version)
count_is_less_than_one    (debugging version)

```

**Warnings**

```

index_out_of_range
count_out_of_range

```

## SEE ALSO

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

To change the set of tag references associated with a shape, use the `GXSetShapeTags` function, described next.

***GXSetShapeTags***

---

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

```

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

```

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

## Shape Objects

**DESCRIPTION**

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

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

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

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

**Warnings**

`index_out_of_range`  
`count_out_of_range`

**Notices (debugging version)**

`tag_already_set`

**SEE ALSO**

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

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

## Directly Manipulating a Shape's Geometry

---

This section describes the functions you use to directly manipulate the geometry of a shape object. Unlike most calls to QuickDraw GX objects, these functions give you direct access to the data of a geometry—in QuickDraw GX memory—without regard to the shape object it is part of. You typically call the functions in this order:

- GXLockShape
- GXGetShapeStructure
- GXUnlockShape
- GXChangedShape

### *GXLockShape*

---

You can use the GXLockShape function to load a shape into memory and lock its geometry into a fixed memory location.

```
void GXLockShape(gxShape target);
```

target      A reference to the shape to be loaded and locked.

#### DESCRIPTION

The GXLockShape function prevents a shape from being relocated. You must set the gxDirectShape attribute of the target shape before calling this function.

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

To directly edit a shape's geometry, call GXLockShape followed by GXGetShapeStructure. After editing, call GXUnlockShape followed by GXChangedShape.

#### SPECIAL CONSIDERATIONS

The GXLockShape function is not related to the gxLockedShape shape attribute. If you set the gxLockedShape attribute, you cannot alter the shape's geometry with functions such as GXSetPoint and GXSetRectangle, described in the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. Setting gxLockedShape has no effect on the direct manipulation of geometry using the calls described here.

In low memory conditions with fragmented memory, QuickDraw GX can unlock locked objects and relocate them. Be careful about making memory-intensive calls after locking an object.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

`out_of_memory`  
`shape_is_nil`  
`graphic_type_does_not_have_a_structure` (debugging version)

**Notices (debugging version)**

`directShape_attribute_set_as_side_effect`

## SEE ALSO

The `GXUnlockShape`, `GXGetShapeStructure`, and `GXChangedShape` functions are described in the following three sections.

Shape attributes are described in the section “Shape Attributes” beginning on page 2-16. To set shape attributes, use the `GXSetShapeAttributes` function, described on page 2-74.

***GXUnlockShape***

---

You can use the `GXUnlockShape` function to allow QuickDraw GX to relocate, compress, or unload a shape that has been locked.

```
void GXUnlockShape(gxShape target);
```

`target`      A reference to the shape to unlock.

**DESCRIPTION**

The `GXUnlockShape` function releases a previously locked shape for relocation or other movement.

To directly edit a shape’s geometry, call `GXLockShape` followed by `GXGetShapeStructure`. After editing, call `GXUnlockShape` followed by `GXChangedShape`. Once you call `GXUnlockShape`, the shape’s geometry may be relocated and a pointer returned by `GXGetShapeStructure` may no longer be valid.

**SPECIAL CONSIDERATIONS**

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

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

shape\_is\_nil

**Notices (debugging version)**

shape\_not\_locked

## SEE ALSO

The GXLockShape function is described in the previous section. The GXGetShapeStructure and GXChangedShape functions are described in the following two sections.

The GXDisposeShape function is described on page 2-55.

***GXGetShapeStructure***

---

You can use the GXGetShapeStructure function to get a pointer to the geometry of a shape object.

```
void *GXGetShapeStructure(gxShape source, long *length);
```

**source**        A reference to the shape object whose geometry you need access to.

**length**        A pointer to a long value. On return, the value specifies the size in bytes of the shape's geometry.

*function result* A pointer to the geometry of the source shape object.

**DESCRIPTION**

The GXGetShapeStructure function determines the size of a shape's geometry and returns a pointer to the geometry in the QuickDraw GX heap. You can use the pointer to examine or change the geometry without copying the geometry into your application's heap and back again.

Before calling GXGetShapeStructure, you should first call GXLockShape to prevent the geometry from being relocated and you should set the gxDirectShape attribute to make the shape accessible in the QuickDraw GX heap. After you are finished examining or changing the geometry, call GXUnlockShape. If you change the shape's geometry, you must call the GXChangedShape function to notify QuickDraw GX that the shape's cache is no longer valid.

To edit a geometry, you need to know its structure. GXGetShapeStructure returns a pointer and a size only; it does not provide you with any information about the internal structure of the geometry. For example, if the source shape is a path, you must cast the function result to a gxPaths pointer. Such information is not described in this book.

## Shape Objects

If you call this function for a shape that has no geometry (shape types `gxEmptyType` and `gxFullType`), the function posts a `graphic_type_has_no_structure` warning.

**SPECIAL CONSIDERATIONS**

If you do not set the `gxDirectShape` attribute or do not lock the shape, QuickDraw GX does them for you as a side effect of the `GXGetShapeStructure` function call. You must still call `GXUnlockShape` to unlock the shape and, if you wish, reset the attribute.

This function is rarely needed. In most instances, you can manipulate a shape's geometry with calls to geometry-specific functions such as `GXGetRectangle` or `GXGetGlyphTangents`. This function is provided as a fast alternative to those functions, but be aware that it may fail in low-memory conditions; see "Special Considerations" under the description of the `GXLockShape` function, on page 2-80.

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

`out_of_memory`  
`shape_is_nil`  
`graphic_type_does_not_have_a_structure` (debugging version)

**Notices (debugging version)**

`lockShape_called_as_side_effect`

**SEE ALSO**

The `GXLockShape` and `GXUnlockShape` functions are described in the previous sections. The `GXChangedShape` function is described in the following section.

Shape types are described in the section "Shape Type" beginning on page 2-9.

Shape geometry structures, and the functions for manipulating them, are described in the shape-specific chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*.

***GXChangedShape***

You can use the `GXChangedShape` function to notify QuickDraw GX that you have directly edited the geometry of a shape.

```
void GXChangedShape(gxShape target);
```

**target**      A reference to the shape object whose geometry you have directly edited.

## Shape Objects

**DESCRIPTION**

The `GXChangedShape` function notifies QuickDraw GX that the geometry of the shape referenced by the `target` parameter has been modified. QuickDraw GX can then use that information to invalidate existing shape caches, if necessary.

You need to call this function only if you have directly edited a shape's geometry by using the pointer returned by the `GXGetShapeStructure` function. If you edit a shape geometry using any other shape-editing function, you do not need to call `GXChangedShape`.

To directly edit a shape's geometry, call `GXLockShape` followed by `GXGetShapeStructure`. After editing, call `GXUnlockShape` followed by `GXChangedShape`.

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

`shape_is_nil`

**SEE ALSO**

The `GXLockShape`, `GXUnlockShape`, and `GXGetShapeStructure` functions are described in the previous sections.

Shape caches are discussed in the section "Caching Shape Objects" beginning on page 2-27.

Other functions for editing shape geometries are described in the shape-specific chapters of *Inside Macintosh: QuickDraw GX Graphics* and *Inside Macintosh: QuickDraw GX Typography*.

## Drawing and Hit-Testing Shapes

---

This section describes the basic QuickDraw GX functions for drawing and hit-testing shapes: `GXDrawShape` and `GXHitTestShape`.

### ***GXDrawShape***

---

You can use the `GXDrawShape` function to draw any shape.

```
void GXDrawShape(gxShape source);
```

`source`      A reference to the shape object of the shape to draw.



## Shape Objects

## DESCRIPTION

The `GXDrawShape` function draws the shape referenced by the source parameter, taking into account the properties specified in the shape's style, ink, and transform objects. It draws the shape to the display device or devices specified indirectly in the shape's transform object.

As part of preparation for drawing, QuickDraw GX makes preliminary calculations and stores the results in caches. You can in some cases speed drawing by having the calculations and cache storage occur ahead of time; you can do that by setting the source shape's `gxCachedShape` attribute or by calling the `GXCacheShape` function.

## ERRORS, WARNINGS, AND NOTICES

In addition to the errors listed below, the `GXDrawShape` function can post font-related errors if it is drawing text.

**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_bitmap_exceeds_implementation_limit</code>	
<code>pattern_lattice_out_of_range</code>	(debugging version)

**Warnings**

<code>character_substitution_took_place</code>	
<code>graphic_type_cannot_be_dashed</code>	(debugging version)
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)
<code>unable_to_draw_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)
<code>face_override_style_font_must_match_style</code>	(debugging version)

## SEE ALSO

The `GXDrawShape` function as applied to geometric shapes, and other functions for drawing geometric shapes, are described in the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. The function as applied to typographic shapes, and other functions for drawing typographic shapes, are described in the text shapes, glyph shapes, and layout shapes chapters of *Inside Macintosh: QuickDraw GX Typography*.

Transform objects and their relation to display devices are described in the chapter "Transform Objects" in this book.

The `gxCachedShape` attribute is described in Table 2-4 on page 2-16. The `GXCacheShape` function is described on page 2-62. The differences between the two caching methods are described in the section "Caching Shape Objects" beginning on page 2-27.

## *GXHitTestShape*

---

You can use the `GXHitTestShape` function to convert a point in space (which may represent, for example, the location of a mousedown event) into a distance from a particular part of the geometry of a shape.

```
gxShapePart GXHitTestShape(gxShape target, const gxPoint *test,
                           gxHitTestInfo *result);
```

<code>target</code>	A reference to the shape to hit-test.
<code>test</code>	A pointer to a point structure specifying the location to hit-test the shape against. The location must be specified in the local coordinates of the shape.
<code>result</code>	A pointer to a <code>gxHitTestInfo</code> structure. On return, the structure contains detailed information about the hit-test.

*function result* The parts of the shape corresponding to the location specified in the `test` parameter (within the tolerance limits for the hit-test).

### DESCRIPTION

The `GXHitTestShape` function takes a shape reference and a point in geometry or local space and returns whether or not the point was within a certain distance (tolerance) of one of a set of specified parts of the shape. With this function you can, for example, respond to user actions such as mouse clicks or movements by highlighting or selecting parts of shapes. The tolerance and the shape parts are defined in the hit-test parameters of the shape's transform object. The function returns the shape parts that were hit, or else the value `gxNoPart` if no tested part of the shape was hit.

On return, the `result` parameter contains a filled-out `gxHitTestInfo` structure. Only the first three fields are filled out by `GXHitTestShape`:

- The `what` field describes the shape parts that were hit, if any. It is identical to the function result from this function.
- The `index` field identifies, by (1-based) index, the nearest point in the geometry to the hit point.
- The `distance` field describes the distance from the hit point to the closest point on the shape part that was hit. (If no part was hit, this value is undefined.) If more than one shape part was hit, this is the distance to the first shape part encountered that was within the tolerance of the hit point. The order in which shape parts are examined during hit-testing is defined by the `gxShapeParts` enumeration.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

Because it performs many of the calculations involved in drawing, the `GXHitTestShape` function can post, in addition to the errors listed below, any errors and warnings associated with the `GXDrawShape` function. Therefore `GXHitTestShape` can post font-related errors if it is caching text.

**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_bitmap_exceeds_implementation_limit</code>	
<code>pattern_lattice_out_of_range</code>	(debugging version)

**Warnings**

<code>character_substitution_took_place</code>	
<code>graphic_type_cannot_be_dashed</code>	(debugging version)
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)
<code>unable_to_draw_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)
<code>face_override_style_font_must_match_style</code>	(debugging version)

## SEE ALSO

Hit-testing is discussed in the section “Drawing and Hit-Testing Shapes” beginning on page 2-35.

The `gxHitTestInfo` structure is described on page 2-50.

The `gxShapeParts` enumeration and the `gxShapePart` mask are also described in the hit-test parameters section of the chapter “Transform Objects” in this book.

## Flattening and Unflattening Shape Objects

The two functions described in this section allow you to convert shapes into a compressed, stream-based format for storage or transmission, and to reconstruct shapes from their compressed form.

## *GXFlattenShape*

---

You can use the `GXFlattenShape` function to convert the object form of a shape—including all the objects that it references—into a stream format that is public and suitable for storage and parsing.

```
void GXFlattenShape(gxShape source, gxFlattenFlag flags,
                   struct gxSpoolBlock *block);
```

<code>source</code>	A reference to the shape you wish to flatten.
<code>flags</code>	A set of flags that specify whether or not to save additional information with the flattened file.
<code>block</code>	A pointer to a spool block structure. QuickDraw GX uses information in the spool block to create and store the flattened data.

### DESCRIPTION

The `GXFlattenShape` function creates a flattened version of the shape referenced by the `source` parameter and places it into a buffer pointed to by the spool block specified in the `block` parameter.

Before calling `GXFlattenShape`, you need to allocate a spool block structure and a buffer to hold the flattened data, and place a pointer to the buffer and a specification of its size into the spool block. You also place into the spool block a pointer to an application-defined spool function that writes the flattened data from the buffer to a file. The spool function responds to commands from QuickDraw GX to open, write, and close the file used to hold the flattened data.

If your spool block structure specifies `nil` for the buffer pointer and 0 for its size, QuickDraw GX allocates a default buffer (512 KB in version 1.0 of QuickDraw GX) for you.

Upon completion of the function, QuickDraw GX writes into the spool block the number of bytes of flattened data it has placed into the buffer. It also writes other information into the spool block; your spool function can use that information if you want it to parse the flattened file as flattening occurs. Normally, however, for simple flattening of shapes, your application need not read any of the information returned in the spool block, and your spool function needs to read only the size of the flattened data in the buffer.

Note that flattening a shape causes flattened versions of all its referenced objects, such as its style, ink, and transform—and all of their referenced objects in turn—to be stored as well. To flatten a group of shapes, place them in a picture and flatten the picture.

If you set the `gxFontListFlatten`, `gxFontGlyphsFlatten`, or `gxFontVariationsFlatten` flag in the `flags` parameter when calling this function, `GXFlattenShape` creates a tag object and attaches it to the source shape. The tag object is of type `'flst'` and lists the names of the fonts referenced in the shape, the individual glyphs used in the shape, or the descriptions of any font variations used in the shape, respectively.

## Shape Objects

If you set the `gxBitmapAliasFlatten` flag in the `flags` parameter when calling this function, `GXFlattenShape` includes with the flattened shape all image data from any bitmap shapes that are referenced by the shape. If this flag is not set, image data from bitmap shapes whose image data is disk-based is not included in the flattened shape. That image data is not lost, however, because a tag object specifying the file holding the image data is flattened along with the shape.

The flattened stream created by `GXFlattenShape` consists of a series of opcodes and associated data, following the QuickDraw GX stream format.

**SPECIAL CONSIDERATIONS**

If the source shape already has a tag object of type `'flst'` attached to it, `GXFlattenShape` replaces that tag with a new tag of type `'flst'`; it also posts a `tags_of_type_flst_removed` warning.

If the `block` parameter is `nil`, this function returns a `parameter_is_nil` error. If the `spool-function` pointer in the `spool` block passed in the `block` parameter is `nil`, this function returns a `spoolProcedure_is_nil` error. If the `spool` function signals an error during either flattening or unflattening, QuickDraw GX posts an `unflattening_interrupted_by_client` error. If the `spool` function attempts to call `GXFlattenShape`, QuickDraw GX posts a `procedure_not_reentrant` error.

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

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>procedure_not_reentrant</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>spoolProcedure_is_nil</code>	
<code>unflattening_interrupted_by_client</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)

**Warnings**

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

**SEE ALSO**

The `spool` block structure is described on page 2-49. The format for the application-defined `spool` function is described on page 2-91.

The format for the flattened data, including all opcodes, is described in the stream format chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To convert a flattened shape back to its object-based format, use the `GXUnflattenShape` function, described in the next section.

## *GXUnflattenShape*

---

You can use the `GXUnflattenShape` function to restore the object form of a shape—including all the objects that it references—from a stream-based description created by the `GXFlattenShape` function.

```
gxShape GXUnflattenShape(struct gxSpoolBlock *block, long count,
                        const gxViewport portList[])
```

<code>block</code>	A pointer to a spool block structure. QuickDraw GX uses information in the spool block to unflatten the data.
<code>count</code>	The number of view ports in the view port list (the number of elements in the <code>portList</code> array).
<code>portList</code>	An array of references to view port objects. It is the list of view ports to assign to the transform object for the unflattened shape.

*function result* A reference to the newly created (unflattened) shape.

### DESCRIPTION

The `GXUnflattenShape` function reconstructs a shape object and all its associated objects from stream data in a buffer pointed to by the spool block specified in the `block` parameter.

Before calling `GXUnflattenShape`, you need to allocate a spool block structure and buffer to hold the flattened data, and place a pointer to the buffer and a specification of its size into the spool block. You also place into the spool block a pointer to an application-defined spool function that reads the flattened data into the buffer. The spool function responds to commands from QuickDraw GX to open, read, and close the file containing the flattened data.

Note that unflattening a shape also causes creation of all its referenced objects, such as its style, ink, and transform, and all of their referenced objects. Unflattening a picture can cause the creation of many shape objects.

The flattened stream as read by `GXUnflattenShape` consists of a series of opcodes and associated data, following the QuickDraw GX stream format.

### SPECIAL CONSIDERATIONS

If no error occurs, the `GXUnflattenShape` function creates one or more QuickDraw GX objects. You are responsible for disposing of those objects when you no longer need them.

If the spool function signals an error during either flattening or unflattening, QuickDraw GX posts an `unflattening_interrupted_by_client` error. If the spool function attempts to call `GXUnflattenShape`, QuickDraw GX posts a `procedure_not_reentrant` error.

## Shape Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out_of_memory	
procedure_not_reentrant	
parameter_is_nil	(debugging version)
spoolProcedure_is_nil	
unflattening_interrupted_by_client	
font_not_found	
parameter_out_of_range	(debugging version)
inconsistent_parameters	(debugging version)

**Warnings**

unrecognized_stream_version
bad_data_in_stream

## SEE ALSO

The spool block structure is described on page 2-49. The format for the application-defined spool function is described on page 2-91.

The format for the flattened data, including all opcodes, is described in the stream format chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To convert a QuickDraw GX shape to its flattened form, use the `GXFlattenShape` function, described in the previous section.

## Application-Defined Spool Function

---

This section describes the interface to an application-defined function that can be used for saving and restoring shape objects.

### *MySpoolProc*

---

The QuickDraw GX functions `GXFlattenShape` and `GXUnflattenShape` require the calling application to supply a pointer to a function that, respectively, saves flattened data or supplies data to be flattened. The flattening/unflattening spool function has the following interface:

```
long MySpoolProc(gxSpoolCommand command,
                 struct gxSpoolBlock *block);
```

**command**      A selector with which QuickDraw GX specifies the operation the spool function is to perform.

**block**          A pointer to the spool block used for the current flattening or unflattening operation.

*function result*   Zero if the unflattening or flattening operation can continue; nonzero if QuickDraw GX must abort the operation.

## Shape Objects

**DESCRIPTION**

The purpose of the flattening/unflattening spool function is to move flattened data into or out of memory as instructed by QuickDraw GX. You place a pointer to the spool function in the appropriate field of the spool block structure that you allocate before calling `GXFlattenShape` or `GXUnflattenShape`.

Your spool function should respond to the `command` parameter and perform the appropriate operation. Constants for the recognized spool commands are defined in the `gxSpoolCommands` enumeration:

Constant	Value	Explanation
<code>gxOpenReadSpool</code>	1	The spool function is to open the flattened file for reading into the buffer used by <code>GXUnflattenShape</code> .
<code>gxOpenWriteSpool</code>	2	The spool function is to open a file for receiving data from the buffer used by <code>GXFlattenShape</code> .
<code>gxReadSpool</code>	3	The spool function is to read the data into the buffer for use by <code>GXUnflattenShape</code> .
<code>gxWriteSpool</code>	4	The spool function is to write the data placed into the buffer by <code>GXFlattenShape</code> to the file.
<code>gxCloseSpool</code>	6	The spool function is to close the file.

Your spool function's function result is a status indicator to QuickDraw GX. If the operation must be aborted (for example, if a file error occurs), return a nonzero value as the function result. Otherwise, return 0 and the flattening or unflattening operation can continue.

For simple flattening and unflattening, your spool function need only read and write the amounts of data specified by QuickDraw GX. However, you can also write a spool function that actually parses the data stream during flattening and unflattening; see Listing 2-5 on page 2-41 for an example.

**SPECIAL CONSIDERATIONS**

During a flattening or unflattening procedure, a memory error can cause a restart of the function, which might not necessarily reset the stream back to its original position. Therefore, your open, read, and write routines must always be sure to set the stream to the correct position in the buffer each time.

**SEE ALSO**

The spool block structure is described on page 2-49.

The `GXFlattenShape` function is described on page 2-88. The `GXUnflattenShape` function is described on page 2-90.



## Summary of Shape Objects

---

### Constants and Data Types

---

#### *The Shape Object*

```
typedef struct gxPrivateShapeRecord *gxShape;
```

#### *Shape Type*

```
enum gxShapeTypes {
    gxEmptyType = 1,      /* empty shape; contained by all geometries */
    gxPointType,          /* point shape */
    gxLineType,           /* line shape */
    gxCurveType,          /* curve shape */
    gxRectangleType,      /* rectangle shape */
    gxPolygonType,        /* polygon shape; can represent multiple polygons */
    gxPathType,           /* path shape; can include lines and curves */
    gxBitmapType,         /* bitmap shape */
    gxTextType,           /* text shape; single size, encoding, and style */
    gxGlyphType,          /* glyph shape; can use multiple text styles */
    gxLayoutType,         /* layout shape; can include linguistic info */
    gxFullType,           /* full shape; includes all geometries */
    gxPictureType         /* picture shape; can contain other shapes */
};
```

```
typedef long gxShapeType;
```

#### *Shape Fill*

```
enum gxShapeFills {
    gxNoFill,
    gxOpenFrameFill,
    gxFrameFill = gxOpenFrameFill,
    gxClosedFrameFill,
    gxHollowFill = gxClosedFrameFill,
    gxEvenOddFill,
    gxSolidFill = gxEvenOddFill,
    gxWindingFill,
    gxInverseEvenOddFill,
```

## Shape Objects

```

    gxInverseSolidFill = gxInverseEvenOddFill,
    gxInverseFill = gxInverseEvenOddFill,
    gxInverseWindingFill
};

```

```

typedef long gxShapeFill;

```

**Shape Attributes**

```

enum gxShapeAttributes {
    gxNoAttributes,
    gxDirectShape          = 0x0001,    /* prefer shape data to be in GX heap */
    gxRemoteShape          = 0x0002,    /* prefer shape data on accelerator */
    gxCachedShape          = 0x0004,    /* pre-calculate to optimize drawing */
    gxLockedShape          = 0x0008,    /* prevent changes to shape's geometry */
    gxGroupShape           = 0x0010,    /* treat as one shape for hit-testing */
    gxMapTransformShape    = 0x0020,    /* alter transform, not shape geometry */
    gxUniqueItemsShape     = 0x0040,    /* copy items added to picture shapes */
    gxIgnorePlatformShape  = 0x0080,    /* assume glyph, not character, codes */
    gxNoMetricsGridShape   = 0x0100,    /* draw without font scaler's hinting */
    gxDiskShape            = 0x0200,    /* unload this shape first */
    gxMemoryShape          = 0x0400     /* unload this shape last */
};

```

```

typedef long gxShapeAttribute;

```

**Flatten Flags**

```

enum gxFlattenFlags{
    gxFontListFlatten      = 0x01,      /* add a tag listing fonts used */
    gxFontGlyphsFlatten    = 0x02      /* add a tag listing glyphs used */
    gxFontVariationsFlatten = 0x04,      /* add a tag for font variations */
    gxBitmapAliasFlatten    = 0x08      /* flatten all bitmap image data */
};

```

```

typedef long gxFlattenFlag;

```

**The Spool Block Structure**

```

struct gxSpoolBlock {
    /* these fields are read from (but not written to) by QuickDraw GX */
    gxSpoolProcPtr    spoolProcedure;    /* pointer to spool function */
    void              *buffer;           /* pointer to application buffer */
    long               bufferSize;        /* bytes for QuickDraw GX to use */
};

```

## Shape Objects

```

/* these fields are written to (but not read from) by QuickDraw GX */
long          count;          /* bytes for app to read/write */
long          operationSize;   /* size including operand byte */
long          operationOffset; /* offset within current operation */
gxGraphicsOpcode lastTypeOpcode; /* type of last created object */
gxGraphicsOpcode currentOperation; /* last op. emitted or interpreted */
gxGraphicsOpcode currentOperand; /* such as gxTransformTypeOpcode */
unsigned char  compressed;     /* a gxTwoBitCompressionValues */
};

```

***The Hit-Test Info Structure***

```

struct gxHitTestInfo {
    gxShapePart    what;
    long           index;
    Fixed          distance;
    gxShape        which;
    gxShape        containerPicture;
    long           containerIndex;
    long           totalIndex;
};

```

***Spool Commands***

```

enum gxSpoolCommands {
    gxOpenReadSpool= 1,
    gxOpenWriteSpool,
    gxReadSpool,
    gxWriteSpool,
    gxCloseSpool,
};

typedef long gxSpoolCommand;

```

**Functions*****Creating and Manipulating Shape Objects***

```

gxShape GXGetDefaultShape    (gxShapeType aType);
void GXSetDefaultShape       (gxShape target);
gxShape GXNewShape           (gxShapeType aType);
void GXDisposeShape          (gxShape target);
long GXGetShapeSize          (gxShape source);

```

## Shape Objects

```

gxShape GXCopyToShape      (gxShape target, gxShape source);
gxShape GXCopyDeepToShape  (gxShape target, gxShape source);
boolean GXEqualShape       (gxShape one, gxShape two);
gxShape GXCloneShape       (gxShape source);
void GXCacheShape          (gxShape source);
void GXDisposeShapeCache   (gxShape target);
long GXGetShapeCacheSize   (gxShape source);

```

***Manipulating Shape Object Properties***

```

gxShapeType GXGetShapeType (gxShape source);
void GXSetShapeType        (gxShape target, gxShapeType newType);
void GXSetShapeGeometry    (gxShape target, gxShape geometry);
gxShapeFill GXGetShapeFill (gxShape source);
void GXSetShapeFill        (gxShape target, gxShapeFill newFill);
gxStyle GXGetShapeStyle    (gxShape source);
void GXSetShapeStyle        (gxShape target, gxStyle newStyle);
gxInk GXGetShapeInk        (gxShape source);
void GXSetShapeInk         (gxShape target, gxInk newInk);
gxTransform GXGetShapeTransform
                           (gxShape source);
void GXSetShapeTransform   (gxShape target, gxTransform newTransform);
gxShapeAttribute GXGetShapeAttributes
                           (gxShape source);
void GXSetShapeAttributes  (gxShape target, gxShapeAttribute attributes);
void GXResetShape          (gxShape target);
long GXGetShapeOwners      (gxShape source);
long GXGetShapeTags        (gxShape source, long tagType, long index,
                           long count, gxTag items[]);
void GXSetShapeTags        (gxShape target, long tagType, long index,
                           long oldCount, long newCount,
                           const gxTag items[]);

```

***Directly Manipulating Shape Geometry***

```

void GXLockShape          (gxShape target);
void GXUnlockShape        (gxShape target);
void *GXGetShapeStructure (gxShape source, long *length);
void GXChangedShape       (gxShape target);

```

***Drawing and Hit-Testing Shapes***

```

void GXDrawShape          (gxShape source);
gxShapePart GXHitTestShape (gxShape target, const gxPoint *test,
                             gxHitTestInfo *result);

```

***Flattening and Unflattening Shapes***

```

void GXFlattenShape       (gxShape source, gxFlattenFlag flags,
                             gxSpoolBlock *block);
gxShape GXUnflattenShape   (struct gxSpoolBlock *block, long count,
                             const gxViewPort portList[]);

```

**Application-Defined Spool Function**

---

```

long MySpoolProc          (gxSpoolCommand command,
                             struct gxSpoolBlock *block);

```

