

Glyph Shapes

This chapter describes the QuickDraw GX glyph shape and its contents, and tells how to create and use a glyph shape. Read this chapter if you want your application to use a typographic shape as a graphic entity—for example, to dash text along a path or place glyphs in arbitrary positions.

Before reading this chapter, you should be familiar with the information in the chapters “Introduction to QuickDraw GX Typography” and “Typographic Shapes” in this book. You should also be familiar with the information in *Inside Macintosh: QuickDraw GX Objects*.

This chapter discusses the QuickDraw GX glyph shape and describes the properties of glyph shapes. It then shows how to use QuickDraw GX functions to

- create and draw glyph shapes
- change parts of a glyph shape
- set the tangents, positions, and advance bits arrays of a glyph shape

About Glyph Shapes

The **glyph shape** is a typographic shape that allows you to vary the position, font, rotation, and scale of each glyph in a line of text. The glyph shape has the same set of properties as other QuickDraw GX shape objects. Its shape type is `gxGlyphShape`, and its geometry is unique.

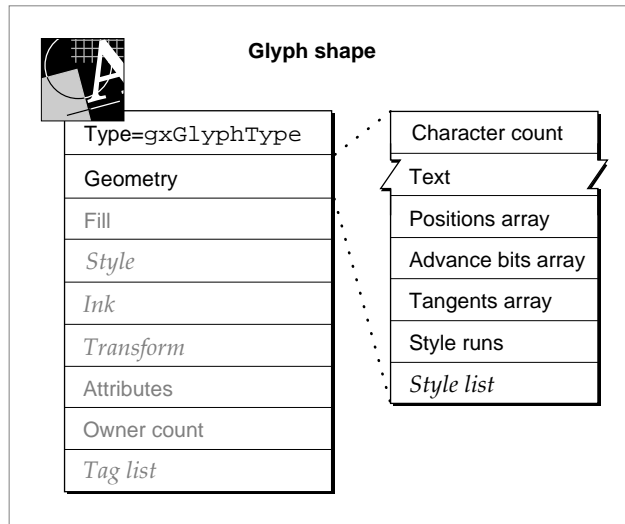
A glyph shape is drawn as one or more glyphs. One glyph shape can represent a character, such as “a”, or a ligature, such as “fi”. Note that the glyph shape, as defined in QuickDraw GX, is a specific type of shape object. In this context, *shape* does not refer to the form of an individual glyph.

The glyph shape is primarily useful as a graphic entity. Because of its geometry, the glyphs in a glyph shape can be positioned individually, yet still be related to the other glyphs in the shape. This allows you to create a number of graphic effects with the glyph shape that you can’t with the text or layout shapes, such as setting text along an arbitrary path—for example, in a circle.

The glyph shape displays only the glyphs for the specific character codes or glyph codes it contains. For example, if you draw a glyph shape that contains adjacent character codes for the glyphs “f” and “i”, the glyph shape does not automatically display an “fi” ligature. If you need such automatic linguistic processing of glyphs, you should use the layout shape, described in the chapter “Layout Shapes” in this book.

The Geometry of a Glyph Shape

The geometry of a glyph shape contains seven elements, as shown in Figure 4-1. Note that, because a glyph shape is an object and not a data structure, the order of the properties as shown in Figure 4-1 is completely arbitrary.

Figure 4-1 Geometry of a glyph shape

The geometry of the glyph shape contains the following elements:

- **Character count.** The number of characters in the text array. Note that the number of characters is not necessarily the same as the number of bytes in the text array.
- **Text.** An array of character codes or glyph codes. Whether each character code is 8-bit or 16-bit depends on the platform in the glyph shape's style object and on the actual byte values. If the value of the shape attribute `gxIgnorePlatformShape` is clear, then the text is interpreted as an array of character codes. If it is set, it is interpreted as an array of 16-bit glyph codes. (See the chapter "Font Objects" in this book for more information about platforms, character codes, and glyph codes.)
- **Positions array.** Contains positions for the origin of each character or glyph in the shape. These positions can be relative to the advance width of the previous character or glyph, or they can be absolute positions in geometry coordinates. The first position in the positions array marks the origin of the first character or glyph in the shape.
- **Advance bits array.** Determines whether the points in the positions array are absolute or relative. The advance bits array contains 1 bit for every character or glyph in the shape.
- **Tangents array.** Determines the scaling and orientation of the characters or glyphs in the shape. It contains one entry for each character or glyph in the shape.
- **Style runs array.** Determines how many characters or glyphs in the shape each style applies to. Each entry is the number of characters or glyphs to which the equivalent entry in the style list applies.
- **Style list.** An array of references to the style object attached to the shape. This allows a glyph shape to have more than one style. If there are styles in this array, QuickDraw GX does not use the information in the style object associated with the shape.

You use the functions described in this chapter to set the values in the geometry of the glyph shape.

The Positions and Advance Bits Arrays

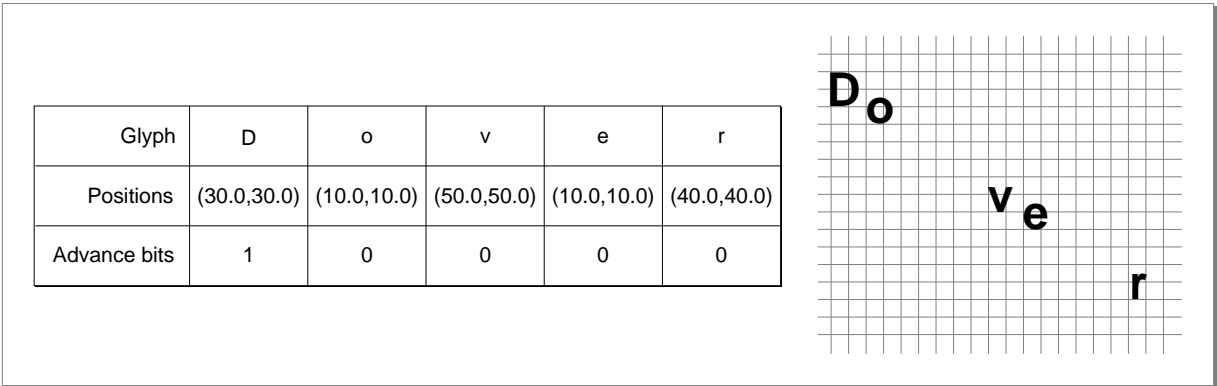
There are two ways of positioning a glyph in a glyph shape: it can be positioned relative to the preceding glyph, or it can have its own position—except for the first glyph, which has to be absolute, not relative—specified as a QuickDraw GX coordinate. Each position is stored in the **positions array** as a point, such as (30.0,50.0).

For every position in the positions array, there is a corresponding bit in the **advance bits array**. A value of 0 for an advance bit indicates a relative position—that is, the position of the glyph at that index is relative to the end of the advance width of the preceding glyph. A value of 1 indicates an **absolute position**, which specifies that the position is absolute in geometry coordinates. For example, suppose the positions array specifies the point (30.0,50.0) for a glyph. If the advance bits array indicates a **relative position**, QuickDraw GX sets the origin of the glyph at 30 points along the x-axis and 50 points along the y-axis from the end of the advance width of the previous glyph. If the advance bits array indicates that the position is absolute, QuickDraw GX draws the glyph at the point (30.0,50.0) in geometry space.

The order of bits in the array moves from high to low, meaning that the highest-order bit in the entire array is for glyph 1. If there are less than 32 bits per glyph shape, the low-order bits are ignored. Because the advance bits are stored in long integers, the array contains a minimum of 32 bits per glyph shape. The value of the first bit is always 1, because the first position in the array is always absolute.

In Figure 4-2, the glyphs in the shape “Dover” have the absolute position (30.0,30.0) and the relative positions (10.0,10.0), (50.0,50.0), (10.0,10.0), and (40.0,40.0). The initial “D” has an absolute position; the glyphs following have relative positions, because their corresponding bits in the advance bits array are 0.

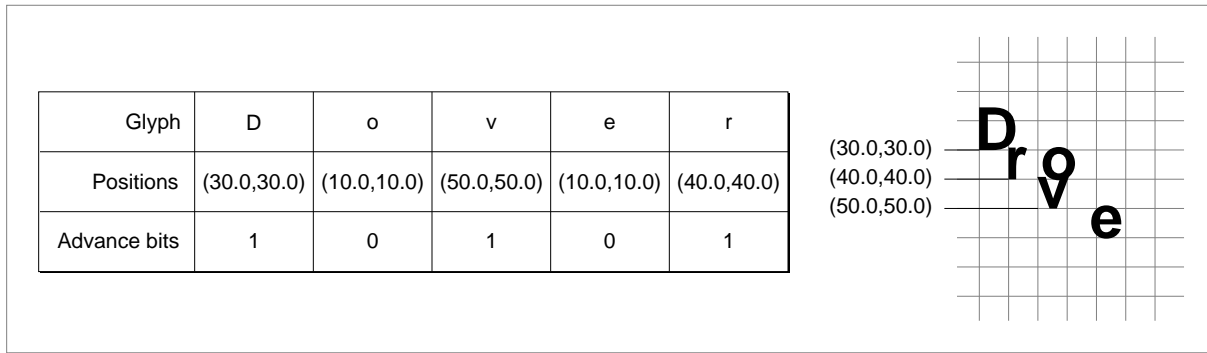
Figure 4-2 The effect of the positions and advance bits arrays on glyph placement



Glyph Shapes

In Figure 4-3, the shape has the same glyphs and the same values in the positions array. However, the first, third, and fifth glyphs now have absolute positioning.

Figure 4-3 The same shape with a new advance bits array



If you don't set positions for the glyph shape explicitly, QuickDraw GX sets the values of all the advance bits (except the first one) and positions to 0 and (0.0,0.0) respectively, resulting in glyphs that line up next to one another, each starting where the previous glyph's advance width ends. It sets the first advance bit to 1, because the first position is always absolute.

However, if you set positions for the shape without specifying the advance bits, QuickDraw GX sets every bit in the advance bits array to 1, because QuickDraw GX interprets all of the positions as absolute positions. If you want some positions to be relative, you must pass the advance bits array with the appropriate bits set to 0.

The Tangents Array

Each glyph in a glyph shape may have an optional **tangent** vector that specifies the scale and angle that QuickDraw GX should apply before drawing that glyph. The tangent vector is represented by a point: the direction of the vector is the direction from the point (0.0,0.0) to the tangent point, and the length of the vector is determined by the length of the line between those two points. The glyph with that tangent is laid out along that vector accordingly, with its glyph origin aligned with the point (0.0,0.0) and the advance width scaled to match the length of the vector. For a 1-unit-long vector, the glyph is "scaled" to 100 percent of its size.

Here are some characteristics of a tangent that you should know about:

- The tangent vector coordinate system matches the QuickDraw GX coordinate system: values for the tangent are always (x,y), regardless of the baseline of the glyph or glyphs associated with that tangent.
- QuickDraw GX always processes the characters in storage order and always draws the text of a typographic shape from left to right (regardless of the inherent line direction of the text), so the direction of the tangent is the direction of the advance vector.

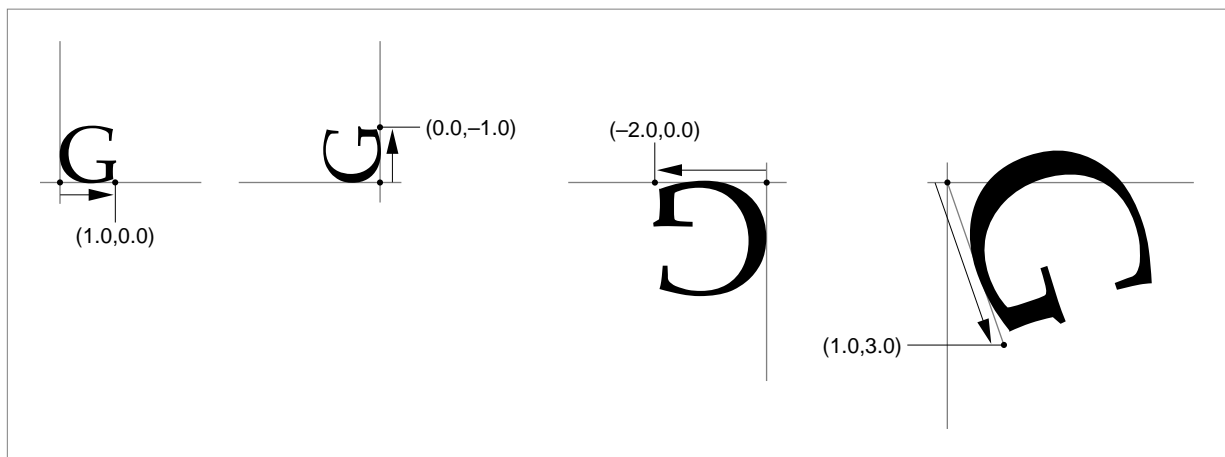
Glyph Shapes

Figure 4-4 shows the effect of a variety of tangents on a glyph. All of the glyphs shown have the same style, and therefore the same font and text size; they differ only in how the tangents affect their appearance.

In the first example, a glyph has the default tangent vector of $(1.0,0.0)$, which specifies that there is a scaling factor of 1 and no rotation from the natural text direction; a glyph with this tangent vector (typical for Roman text) would draw left to right, and a glyph with the `gxVerticalText` text attribute (typical for Kanji) set in the shape's style would draw top to bottom.

The second example shows a tangent vector of $(0.0,-1.0)$: the glyph is rotated 90 degrees upward from the natural orientation. The third example shows a tangent vector of $(-2.0,0.0)$: notice how the glyph scales to double its normal size to fit the length of the vector. In the fourth example, the glyph follows a tangent of $(1.0,3.0)$. The glyph scales in all directions to fit the tangent.

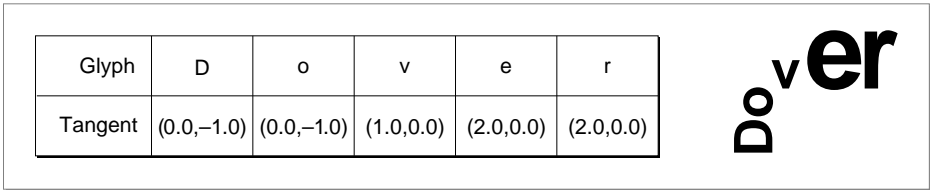
Figure 4-4 Various tangents



In Figure 4-5, the shape “Dover” has five separate tangents, one for each glyph in the shape. The first two rotate the glyph counterclockwise by 90 degrees off the natural baseline. The last three put the shape back onto the natural baseline: note how the origin of the glyph “v” begins where the advance width of the glyph “o” ends, even though the glyphs have different orientations. The last two tangents are $(2.0,0.0)$, which scale those glyphs to twice their given point size. This shape uses the default positions and advance bits, and a single style for the entire shape, allowing the glyph shape to have glyphs of different sizes in a single style run.

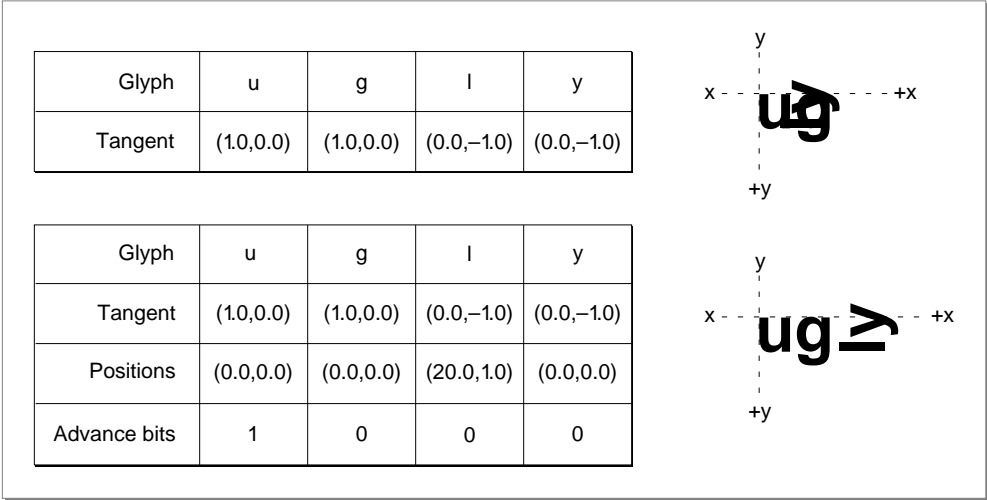
Glyph Shapes

Figure 4-5 The effect of the tangents array on glyph placement



QuickDraw GX begins drawing the glyph origin of a glyph in relation to the end of the advance width of the previous glyph, if there are no other factors affecting the placement of the glyphs in the geometry. In the top part of Figure 4-6, the glyph origin of the glyph “l” begins where the advance width of the glyph “g” ended, and the orientation of the tangents of the glyphs “l” and “y” cause those glyphs to overwrite the previous glyphs. In the bottom part of Figure 4-6, use of the positions and advance bits arrays changes the glyph origins of “l” and “y” so that they are farther away from previous glyphs and avoid overwriting.

Figure 4-6 Tangents used with and without positions



The Style Runs and Style List

Every glyph in a glyph shape must belong to a style run. A **style run** consists of a number of consecutive glyphs that share an associated style object (which contains the font, the typestyle, the text size, and other specific characteristics). By default, QuickDraw GX assigns the style object associated with the shape to all glyphs in a glyph shape. However, each glyph in a glyph shape can be in its own style. The shape’s geometry contains the style list, which is a list of references to style objects. (For more information about what is contained in the style object that applies to typographic shapes, see the chapter “Typographic Styles” in this book.)

Glyph Shapes

Figure 4-7 shows a four-glyph shape with two style runs, each containing two glyphs. The first style run contains the font 24-point Courier Roman for two glyphs, “e” and “r”; the second contains the font 10-point Helvetica® Bold for another two glyphs, “g” and “o”. The style object associated with this shape—that is, the object referenced in the style property of the shape—may contain completely different properties. However, because there are style runs and a style list in the geometry of this glyph shape, QuickDraw GX ignores the information in the “regular” associated style object—the one referenced in the style property of the shape object.

Figure 4-7 The effect of style runs on the appearance of glyphs in a glyph shape



Figure 4-8 shows a glyph shape in which each glyph has its own style. You cannot get the same result using multiple glyph shapes, because the position of each glyph would differ depending on the device characteristics (which affect such values as the advance widths). Each glyph can't be positioned individually to get the same results over many display devices, because device resolutions may vary.

Figure 4-8 An example of a glyph shape with a style run for each glyph



You can use glyph shapes for clipping, dashing, and patterns only if they are in primitive form. (For more information about primitive forms, see the chapter “Typographic Shapes” in this book.) Remember, however, that any shape you want to convert to primitive form cannot have any styles, whether in the style object or in the style list, that contain caps, dashes, patterns, joins, font variations, or text faces.

To pattern a glyph shape, for example, you must include, in the glyph shape's style, a text face that is patterned, or that includes joins, start and end caps, dashing, and so on. (Text faces are described in the chapter “Typographic Styles.”)

You can convert a glyph shape into a text shape, a layout shape, or any other type of shape by using the `GXSetShapeType` function. For more information about `GXSetShapeType`, see the “Shape Objects” chapter of *Inside Macintosh: QuickDraw GX Objects*.

The Default Glyph Shape

The default glyph shape has no text and no starting position (0,0); all of its components are either zero or nil. Like the default text and layout shapes, the default glyph shape has a `gxWindingFill` type.

The default glyph shape has a style and that style is described in the “Typographic Styles” chapter in this book. The default settings for all typographic shapes are described in the chapter “Typographic Shapes” in this book.

Using Glyph Shapes

This section describes how to perform typical operations with the glyph shape. It tells how to

- create and draw a glyph shape
- get information from a glyph shape
- change parts of a glyph shape by modifying the text in the shape and replacing styles and style runs
- manipulate the positions and advance bits arrays to change the position of the shape in the view port
- change the tangent array

Creating and Drawing a Glyph Shape

To create a glyph shape, you use the `GXNewGlyphs` function (described on page 4-22). You can also use the `GXNewShape` function, which is described in the “Shape Objects” chapter of *Inside Macintosh: QuickDraw GX Objects*.

If you create a glyph shape, the shape has one style run (because there is no style list), one absolute position (the starting position of the first glyph of the shape), and the same tangent for each of the glyphs in the shape.

If you want to change the styles in the shape or add new styles to the existing ones, you must determine how many glyphs are in each style run. Listing 4-1 shows how to use the `GXNewGlyphs` function to create a glyph shape with five glyphs in it, with the first two in 30-point New York and the other three in 60-point Geneva.

The code in Listing 4-1 creates the two styles and uses the `GXSetStyleTextSize` and `GXSetStyleFont` functions and the `GXFindFonts` library function to set up each style. It then uses the `GXNewGlyphs` function to create the glyph shape, and positions the shape using the `GXMoveShapeTo` function. (You can also set the position of the shape in the positions array by using `GXSetGlyphPositions` function, described on page 4-33.) It then uses `GXDrawShape` to draw the glyph shape and `GXDisposeShape` to dispose of the shape.

Listing 4-1 Creating a glyph shape with style runs

```

gxShape  myShape;
gxStyle  myStyles[2];
gxFont   newYorkFont = 0, genevaFont = 0;
static const unsigned char myString[] = "glyph";
short    myStyleRuns[2];

/* create two styles */
myStyles[0] = GXNewStyle();
myStyles[1] = GXNewStyle();

/* set up first style as 30-pt.New York*/
GXSetStyleTextSize(myStyles[0], ff(30));
GXFindFonts(0,gxFullFontName,gxMacintoshPlatform,gxRomanScript,
gxEnglishLanguage, 8, "New York", 1, 1, &newYorkFont);
GXSetStyleFont(myStyles[0], newYorkFont);
myStyleRuns[0] = 2; //length of text in style

/* set up second style as 60-pt.Geneva */
GXFindFonts(0,gxFullFontName,gxMacintoshPlatform,gxRomanScript,
gxEnglishLanguage, 6, "Geneva", 1, 1, &genevaFont);
GXSetStyleFont(myStyles[1], genevaFont);
GXSetStyleTextSize(myStyles[1], ff(60));
myStyleRuns[1] = 3; //length of text in style

/* create glyph shape using myString and the new styles */
myShape = GXNewGlyphs(sizeof(myString) -1, myString,
                      nil, nil, nil, myStyleRuns, myStyles);

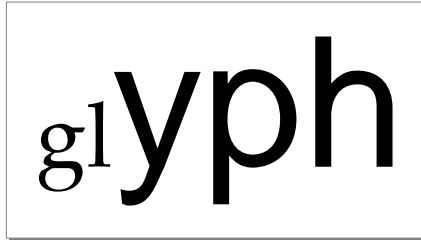
/* move the shape to (50,125)*/
GXMoveShapeTo(myShape, ff(50), ff(125));

GXDrawShape(myShape);

GXDisposeStyle(myStyles[0]);
GXDisposeStyle(myStyles[1]);
GXDisposeShape(myShape);

```

Listing 4-1 produces the output shown in Figure 4-9.

Figure 4-9 A glyph shape with two styles

For more information about `GXSetStyleTextSize` and `GXSetStyleFont` see the chapter “Typographic Styles” in this book. For more information about `GXMoveShapeTo`, `GXDrawShape`, and `GXDisposeShape`, see *Inside Macintosh: QuickDraw GX Objects*.

There are two ways to draw a glyph shape or its equivalent. You can draw an existing glyph shape using the `GXDrawShape` function, described in *Inside Macintosh: QuickDraw GX Objects*. You can also use the `GXDrawGlyphs` function, described on page 4-24, which allows you to draw a glyph shape without first creating the shape.

Getting Information From a Glyph Shape

You can retrieve the information that is in the geometry of a glyph shape—the character or glyph codes, positions, advance bits, tangents, style runs, and style lists—using `GXGetGlyphs`.

Listing 4-2 shows one way of allocating the required space and retrieving all of the glyph shape’s data. You call `GXGetGlyphs` twice—once to find out the number of elements in the shape, and once to retrieve the information. The code first calls `GXGetGlyphs` to retrieve the shape’s byte count, character count, and style-run count. The code must then assign enough space to the variables to hold that data. The code calls `GXGetGlyphs` again to retrieve the information from the glyph shape.

Listing 4-2 Getting all of the information from a glyph shape

```
long          myByteCount, myCharCount, myRunCount;
unsigned char *myText;
gxPoint       *myPositions, *myTangents;
long          *myAdvanceBits;
short         *myStyleRunLengths;
gxStyle       *myStyleRunStyles;

/* get the byte count, character count, and style run count */
myByteCount = GXGetGlyphs(myGlyphsShape, &myCharCount, nil, nil,
                          nil, &myRunCount, nil, nil);
```

Glyph Shapes

```

/* calculate space needed, and assign it to variables */
myText = (unsigned char *) NewPtr(myByteCount);
myPositions = (gxPoint *) NewPtr(myCharCount * sizeof(gxPoint));
myAdvanceBits = (long *) NewPtr(((myCharCount + 31) / 32) *
                                sizeof(long));
myTangents = (gxPoint *) NewPtr(myCharCount * sizeof(gxPoint));
myStyleRunLengths = (short *) NewPtr(myRunCount * sizeof(short));
myStyleRunStyles = (gxStyle *) NewPtr(myRunCount *
                                       sizeof(gxStyle));

/* call GXGetGlyphs again to retrieve the information */
GXGetGlyphs(myGlyphsShape, nil, myText, myPositions,
            myAdvanceBits, myTangents, nil, myStyleRunLengths,
            myStyleRunStyles);

```

You can use the `GXGetGlyphParts` function to retrieve specified glyphs from the source shape. The `GXGetGlyphs` function is described on page 4-26. The `GXGetGlyphParts` function is described on page 4-29.

Changing Parts of a Glyph Shape

You can change any of the information in the geometry of a glyph shape—the text, the style runs, the positions array, the advance bits array, or the tangents array—using the `GXSetGlyphs` or `GXSetGlyphParts` function. The difference between these two functions is that the `GXSetGlyphs` function replaces an entire element of the glyph shape's geometry (for example, the entire positions array), whereas the `GXSetGlyphParts` function allows you to replace a portion of an element (for example, to add some new positions to an existing positions array without changing any of the original data).

To change all or part of the positions array and advance bits array, use the `GXSetGlyphPositions` function, although you must be sure to correlate the data in the new positions array with the values in the advance bits array. The `GXSetGlyphPositions` function is described on page 4-33.

To change the tangents array only, use the `GXSetGlyphTangents` function, which is described on page 4-35.

In each of these functions, if you do not want to change the values for an array, set its parameter to `nil`. If you want to explicitly delete the values for one array and reset its values to the default settings, you must use the constant `gxSetToNil` for that array's parameter.

Changing Text in a Glyph Shape

You can change text in a glyph shape using the `GXSetGlyphs` or `GXSetGlyphParts` function. Use the former if you want to replace all of the text in the shape. (For more information about `GXSetGlyphs`, see page 4-27.)

Glyph Shapes

However, for most editing operations you may want to perform on the shape—whether adding text, deleting text, or replacing text—you should use the `GXSetGlyphParts` function, which is described on page 4-30. Table 4-1 lists some of the parameter settings for changing text in a glyph shape using this function.

Table 4-1 Changing text in a glyph shape using the `GXSetGlyphParts` function

Action	Index	Old character count	New character count	Text
Inserting new text	Glyph index at which new text should start or <code>gxSelectToEndif</code> you want to insert at the end	0	Length of the new text	Pointer to the new text
Replacing and deleting some text	Glyph index at which the new text should start	Number of glyphs to delete from the original shape or <code>gxSelectToEnd</code>	Length of the new text	Pointer to the new text
Replacing all text in the shape	1	<code>gxSelectToEnd</code>	Length of the new text	Pointer to the new text
Deleting all text from the shape	1	<code>gxSelectToEnd</code>	0	<code>nil</code>

Listing 4-3 shows how to create a glyph shape that reads “Shape”. It then uses the `GXSetGlyphParts` function to insert new text, “hips”, into the existing shape, changing the shape so that it reads “Shipshape”.

Listing 4-3 Inserting text into an existing glyph shape

```

char      *myString = "Shape";
char      *newString = "hips";
short     textLength;
gxShape   myShape;

textLength = strlen(myString);
myShape = GXNewGlyphs(textLength, (unsigned char *)myString,
                      nil, nil, nil, nil, nil);
GXMoveShapeTo(myShape, ff(50), ff(50));
GXDrawShape(myShape);

textLength = strlen(newString);

```

Glyph Shapes

```
GXSetGlyphParts(myShape, 2, 0, textLength,
                (unsigned char *)newString,
                nil, nil, nil, nil, nil);
```

```
GXMoveShapeTo(myShape, ff(50), ff(100));
GXDrawShape(myShape);
GXDisposeShape(myShape);
```

Remember that if you add text to a shape in which you have style runs, you must update the runs to reflect the new number of glyphs in the shape.

Changing the Style List and Style Runs Array

When you first create a glyph shape, QuickDraw GX gives it a single style run and the default glyph shape's style. However, you can change the styles of glyphs in the shape when you create the shape, or you can use the `GXSetGlyphParts` function.

Keep in mind that when you change the style list in a glyph shape, you must always pass appropriate values for the style runs as well, even if the style runs are not changing. Likewise, you must always pass values for the style list when you want to change the style runs.

For example, Listing 4-4 shows code that swaps, but doesn't actually change, two typefaces. It initially creates the shape "Birdy" so that the first two glyphs have the New York font and second three glyphs have the Geneva font, and it later reverses the fonts.

The code in Listing 4-4 first creates references to the typefaces New York and Geneva. It uses the `GXNewStyle` function to create the two styles and uses `GXSetStyleTextSize` to set their text sizes, which will not change. The code then uses `GXSetStyleFont` to set the first two glyphs to New York and the remaining three to Geneva, and creates the glyph shape using `GXNewGlyphs`.

The next series of calls swaps the style runs. Note that in the call to `GXSetGlyphParts`, you must reassign the style runs you assigned with `GXNewGlyphs`, even though the style runs are not changing.

Listing 4-4 Changing the style runs of a glyph shape

```
gxShape  myShape;
char     *myString = "Birdy";
short    myStyleRuns[] = {2,3};
gxStyle  myStyles[2];
short    textLength;
gxFont   newYorkFont, genevaFont;

/* create references to typefaces; creates styles and text sizes*/
newYorkFont = FindPNameFont(gxFullFontName, "\pNew York");
genevaFont = FindPNameFont(gxFullFontName, "\pGeneva");
myStyles[0] = GXNewStyle();
```

Glyph Shapes

```

myStyles[1] = GXNewStyle();
GXSetStyleTextSize(myStyles[0], ff(30));
GXSetStyleTextSize(myStyles[1], ff(60));

/* set styles for original shape */
GXSetStyleFont(myStyles[0], newYorkFont);
GXSetStyleFont(myStyles[1], genevaFont);
textLength = strlen(myString);
myShape = GXNewGlyphs(textLength, (unsigned char *)myString,
                      nil, nil, nil, myStyleRuns, myStyles);

/* replace the original style runs with the new ones */
GXSetStyleFont(myStyles[0], genevaFont);
GXSetStyleFont(myStyles[1], newYorkFont);
GXSetGlyphParts(myShape, 1, gxSelectToEnd, textLength,
               nil, nil, nil, nil, myStyleRuns, myStyles);

GXMoveShapeTo(myShape, ff(50), ff(50));
GXDrawShape(myShape);

GXDisposeShape(myShape);
GXDisposeStyle(myStyles[0]);
GXDisposeStyle(myStyles[1]);

```

The `GXSetGlyphParts` function is described on page 4-30; the `GXNewGlyphs` function is described on page 4-22. For more information about the functions `GXSetStyleTextSize` and `GXSetStyleFont`, see the chapter “Typographic Styles” in this book.

Positioning a Glyph Shape

If you want to position the entire glyph shape to start at a particular QuickDraw GX coordinate as the glyph origin for the first glyph in the shape, you can use the `GXMoveShapeTo` function. This function changes the value in the positions array for the first glyph. For example, if the `gxMapTransform` shape attribute is clear, this call to `GXMoveShapeTo` changes the first glyph’s position in the positions array to (30.0,40.0):

```
GXMoveShapeTo(myGlyphsShape, ff(30), ff(40));
```

However, you can explicitly set the positions using the positions array, together with the advance bits array, to position the glyphs of a glyph shape. The positions in the positions array can be absolute or relative, depending on the corresponding bit in the advance bits array for that glyph. A value of 1 indicates that the position is absolute in the geometry; a

Glyph Shapes

value of 0 indicates that the position is relative to the end of the advance width of the previous glyph. There are some other combinations of positions and advance bits arrays to keep in mind:

- If you don't specify the positions or advance bits, QuickDraw GX sets one absolute position and the advance bits array to 0x80000000, which indicates that that first position is absolute.
- If you only specify the positions and do not specify advance bits, QuickDraw GX sets the advance bits array to 0xFFFFFFFF—that is, it sets each of those positions as absolute positions for the shape.
- You cannot set the advance bits to 0x00000000; the first glyph advance must always be absolute.

To place each glyph of the glyph shape at a different location, use the `GXSetGlyphPositions` function to set the positions array of an existing shape, as demonstrated in Listing 4-5. (You can also use the `GXSetGlyphParts` function, a general-purpose function that allows you to change many of the arrays in a glyph shape at the same time. This function is described on page 4-30.)

The listing uses `GXNewGlyphs` to create a basic glyph shape, which includes the default positions array. It then changes the positions and advance bits arrays of the shape using the `GXSetGlyphPositions` function.

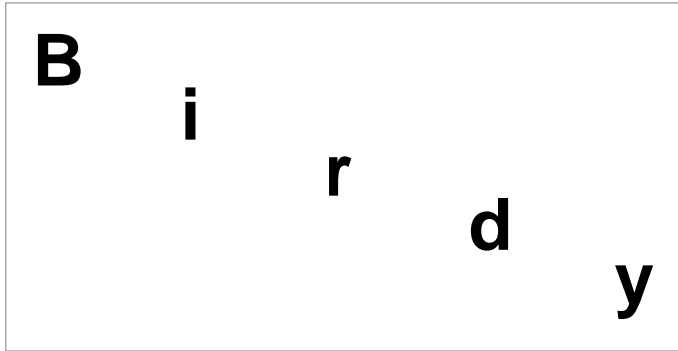
Listing 4-5 Setting the positions and advance bits arrays of a glyph shape

```
gxShape  myShape;
gxPoint  positions[] = { {ff(100), ff(100)}, {ff(50), ff(25)},
                        {ff(50), ff(25)}, {ff(50), ff(25)},
                        {ff(50), ff(25)} };
long      advanceBits[] = {0x80000000};
char      *myString = "Birdy";
short     textLength;

/* create glyph shape, which includes default positions array */
textLength = strlen(myString);
myShape = GXNewGlyphs(textLength, (unsigned char *) myString,
                     nil, nil, nil, nil, nil);
GXSetShapeTextSize(myShape, ff(20));

/* change the shape's positions and advance bits arrays */
GXSetGlyphPositions(myShape, 1, 5, advanceBits, positions);
```

This result of this code is shown in Figure 4-10.

Figure 4-10 A glyph shape with positions and advance bits arrays set

The `GXSetGlyphPositions` function is described on page 4-33; the `GXNewGlyphs` function is described on page 4-22. For more information about `GXSetShapeTextSize`, see the chapter “Typographic Styles” in this book.

Setting the Tangents Arrays

If you want to change the tangents of an existing shape, use the `GXSetGlyphTangents` function, which allows you to replace a single tangent in an existing shape. (You can also use the `GXSetGlyphParts` function, a general-purpose function that is useful if you are replacing much of the information in a glyph shape. This function is described on page 4-30.) For example, the following code draws the third and fifth glyphs of a five-glyph shape at 45-degree angles. (Note that 0.707 represents the square root of 2 divided by 2.)

```
const gxPoint my45DegreeTangent[] = {{f1(0.707), f1(0.707)}};
```

```
GXSetGlyphTangents(myShape, 3, 1, my45DegreeTangent);
```

```
GXSetGlyphTangents(myShape, 5, 1, my45DegreeTangent);
```

Figure 4-11 shows what happens when this code is applied to an existing shape.

Figure 4-11 A glyph shape with 45-degree angle tangents

Glyph Shapes

Of course, because a tangent represents both angle and scale, you can change both values at the same time. (Note that this example had to explicitly design its tangent so that scaling did not occur.) In Listing 4-6, the main function, `GlyphDemo`, uses the subroutine `CreateGlyphTangents` to create a series of tangents that apply varying scales to the glyphs of a glyph shape, a series of tangents that apply varying angles to the glyphs, and then a series of tangents that combine the scales and angles.

Listing 4-6 Creating a series of tangents with varying angles and scales

```
#include "math routines.h"

static void CreateGlyphTangents(long tangentCount,
                                const Fixed angles[],
                                const Fixed scales[],
                                gxPoint *createdTangents)
{
    /* The subroutine uses the angleWalker and scaleWalker variables
    for stepping through the various values for angles and scales sent
    by the main function. The tangentWalker variable stores the
    tangents created as a result of the values given for the angles
    and scales.
    */
    register const Fixed *angleWalker = angles;
    register const Fixed *scaleWalker = scales;
    register gxPoint *tangentWalker = createdTangents;
    gxPolar angleScale = {fixed1, 0};

    /* If there are angles provided, use the next one in the series.
    If there are scales provided, use the next one in the series.
    */
    while (--tangentCount >= 0) {
        if (angleWalker)
            angleScale.angle = *angleWalker++;
        if (scaleWalker)
            angleScale.radius = *scaleWalker++;
        PolarToPoint(&angleScale, tangentWalker++);
    }
}

/* The routine calculates the x and y values of the tangent by
multiplying the scale by the cosine and sine, respectively, of the
angle.
*/

    tangentWalker->x = FractMultiply(x, cos);
    tangentWalker->y = FractMultiply(x, sin);
```

Glyph Shapes

```

        ++tangentWalker;
        --tangentCount;
    }
}

static void GlyphDemo(void)
{
    Fixed angles[11] = { 0, ff(10), ff(15), ff(20), ff(25),
                        ff(30), ff(25), ff(20), ff(15), ff(10),
                        0 };
    Fixed scales[11] = { fl(1.0), fl(1.2), fl(1.4), fl(1.6),
                        fl(1.8), fl(2.0), fl(2.2), fl(2.4),
                        fl(2.6), fl(2.8), fl(3.0) };
    gxPoint tangents[11];
    gxShape myShape;

    /* create tangents that scale each glyph of the shape
    differently but do not affect the angle.
    */
    CreateGlyphTangents(11, nil, scales, tangents);
    myShape = GXNewGlyphs(11, (unsigned char *)"Glyphs Rule", nil,
                        nil, tangents, nil, nil);
    GXSetShapeTextSize(myShape, ff(24));
    GXMoveShape(myShape, ff(10), ff(30));
    GXDrawShape(myShape);

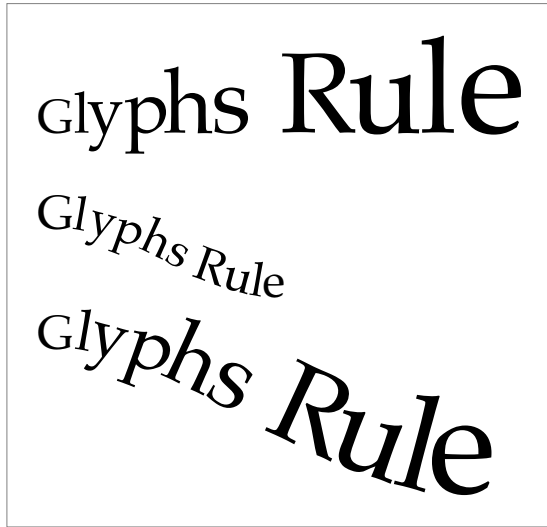
    /* create tangents that draw each glyph of the shape at
    different angles but do not affect the scale or the size of the
    glyphs.
    */
    CreateGlyphTangents(11, angles, nil, tangents);
    GXSetGlyphs(myShape, 0, nil, nil, nil, tangents, nil, nil);
    GXMoveShape(myShape, 0, ff(120));
    GXDrawShape(myShape);

    /* combine the previous two sections: it changes the angle and
    scale of each glyph in the shape. */
    CreateGlyphTangents(11, angles, scales, tangents);
    GXSetGlyphs(myShape, 0, nil, nil, nil, tangents, nil, nil);
    GXMoveShape(myShape, 0, ff(150));
    GXDrawShape(myShape);
    GXDisposeShape(myShape);
}

```

The results of Listing 4-6 are shown in Figure 4-12.

Figure 4-12 Varying the angle and scale of individual glyphs using tangents



The `GXSetGlyphTangents` function is described on page 4-35.

If you are trying to calculate the proper coordinates for a tangent that must approximate a particular angle, you can use the `PolarToPoint` function. You can use the `MapPoints` function to take a single mapping that has the rotation and scaling you desire, plus an array of tangents for your shape, and produce an array of tangents that specify that mapping and rotation.

The `PolarToPoint` and `MapPoints` functions are described the “QuickDraw GX Mathematics” chapter in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Glyph Shapes Reference

Functions

This section describes the functions you use for creating, changing, or setting parts of QuickDraw GX glyph shapes.

Creating and Drawing Glyph Shapes

The `GXNewGlyphs` function creates a new glyph shape. You can also use the `GXNewShape` function to create a glyph shape that has the default style, ink, and transform objects. `GXNewShape` is described in *Inside Macintosh: QuickDraw GX Objects*.

The `GXDrawGlyphs` function draws a string of glyphs in the view port without actually creating a glyph shape. To draw an existing glyph shape, you use the `GXDrawShape`, which is described in *Inside Macintosh: QuickDraw GX Objects*.

GXNewGlyphs

You can use the `GXNewGlyphs` function to create a new glyph shape.

```
gxShape GXNewGlyphs(long charCount, const unsigned char text[],
                    const gxPoint positions[],
                    const long advance[],
                    const gxPoint tangents[],
                    const short styleRuns[],
                    const gxStyle glyphStyles[]);
```

<code>charCount</code>	The number of character codes in the <code>text</code> parameter. The value of <code>charCount</code> must be greater than or equal to 0; if it is not 0, the <code>text</code> parameter must contain an array of character codes. For non-Roman scripts, the byte length may be up to twice the number of character codes.
<code>text</code>	An array of character codes or glyph codes. Whether each character code is 8-bit or 16-bit depends on both the platform in the glyph shape's style object and on the actual byte values. If the value of the glyph attribute <code>gxIgnorePlatformShape</code> is clear, then it returns an array of character codes. If it is set, it returns an array of 16-bit glyph codes.
<code>positions</code>	An array of points, one for each character code in the <code>text</code> parameter. Depending on the corresponding values in the <code>advance</code> bits array, these points are either the relative or absolute positions of the glyphs in geometry space. If you pass <code>nil</code> for this parameter, <code>GXNewGlyphs</code> places the first glyph at (0.0,0.0).
<code>advance</code>	An array of bits, one bit for each character code in the <code>text</code> parameter. The high bit in the long integer corresponds to the first character code in the shape. If the bit is set, then the corresponding position for that glyph in the <code>positions</code> parameter is an absolute position in geometry space. If the bit is not set, the position for that glyph is relative to the end of the advance width of the previous glyph. If you pass <code>nil</code> for both this parameter and the <code>positions</code> parameter, <code>GXNewGlyphs</code> positions each glyph relative according to the advance width of the preceding glyph. If you pass <code>nil</code> for this parameter but not for the <code>positions</code> parameter, <code>GXNewGlyphs</code> gives each glyph an absolute position from the <code>positions</code> array.

Glyph Shapes

<code>tangents</code>	An array of points that serve as tangents. This array determines the scaling or rotation of each glyph. If you pass <code>nil</code> for this parameter, <code>GXNewGlyphs</code> does not apply optional scaling or rotation to each character.
<code>styleRuns</code>	An array of values that specify how many consecutive glyphs use the same style; the styles are stored in the <code>glyphStyle</code> list. The number of entries in the <code>styleRuns</code> array must match the number of entries in the <code>glyphStyles</code> parameter; the sum of the entries in this array should equal the value in the <code>charCount</code> parameter, so that every glyph is part of a style run. The value of each entry must be greater than or equal to 1. If this parameter is <code>nil</code> , <code>GXNewGlyphs</code> uses the default style object for all glyphs in the shape.
<code>glyphStyles</code>	An array of references to style objects. This array should be present only if the style-runs array is present. If you pass <code>nil</code> for this parameter, <code>GXNewGlyphs</code> assigns the style object associated with the glyph shape object to all glyphs in the shape. If an entry in the <code>glyphStyles</code> array is <code>nil</code> , that entry refers to the style object associated with the shape.

function result A reference to the new glyph shape.

DESCRIPTION

The `GXNewGlyphs` function creates a copy of the default glyph shape, sets the owner count of the copy to 1, initializes its geometry with the values in the function's parameters, and returns a reference to the glyph shape as the function result.

The new glyph shape returned by this function contains references to the same style, ink, and transform objects as the default glyph shape.

Most of parameters to `GXNewGlyphs`—`positions`, `advance`, `tangents`, `styleRuns`, and `glyphStyles`—are optional; if you set them to `nil`, QuickDraw GX sets their values to those of the default glyph shape.

The `charCount` parameter indicates the number of character codes present, which may not necessarily equal the number of bytes in the length of the `text` parameter. The interpretation of characters depends on the glyph shape's style attribute.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>missing_parameter</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>count_is_less_than_zero</code>	(debugging version)
<code>parameter_out_of_range</code>	(debugging version)

Glyph Shapes

SEE ALSO

You can also use the `GXNewShape` function, described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*, to create a new glyph shape. The shape type of the glyph shape is `gxGlyphType`. The default glyph shape is described on page 4-10.

To create a new text shape, use the `GXNewText` function, described in the chapter “Text Shapes” in this book.

To create a new layout shape, see the description of the `GXNewLayout` function in the chapter “Layout Shapes” in this book.

For information about the positions and advance bits arrays, see “The Positions and Advance Bits Arrays” on page 4-5. For information about the tangents array, see “The Tangents Array” on page 4-6.

GXDrawGlyphs

You can use the `GXDrawGlyphs` function to draw a string of glyphs without first creating a glyph shape.

```
void GXDrawGlyphs(long charCount, const unsigned char text[],
                  const gxPoint positions[], const long advance[],
                  const gxPoint tangents[], const short styleRuns[],
                  const gxStyle glyphStyles[]);
```

- | | |
|------------------------|--|
| <code>charCount</code> | The number of character codes in the <code>text</code> parameter. For non-Roman scripts, the byte length may be up to twice the number of character codes. This value must be greater than 0, and the <code>text</code> parameter must point at an array of character codes. (If the value of this parameter is 0, the <code>GXDrawGlyphs</code> function does nothing.) |
| <code>text</code> | An array of character codes. |
| <code>positions</code> | An array of points, one for each character code in the <code>text</code> parameter. Depending on the corresponding values in the advance bits array, these points are either the relative or absolute positions of the glyphs in geometry space. If you pass <code>nil</code> for this parameter, <code>GXDrawGlyphs</code> places the first glyph at (0.0,0.0). |
| <code>advance</code> | An array of bits, one bit for each character code in the <code>text</code> parameter. The high bit in the long integer corresponds to the first character code in the shape. If the bit is set, then the corresponding position for that glyph in the <code>positions</code> parameter is an absolute position in geometry space. If the bit is not set, the position for that glyph is relative to the end of the advance width of the previous glyph. If you pass <code>nil</code> for both this parameter and the <code>positions</code> parameter, <code>GXDrawGlyphs</code> positions each glyph at the end of the advance width of the preceding glyph. If you pass <code>nil</code> for this parameter but not for the <code>positions</code> parameter, <code>GXDrawGlyphs</code> gives each glyph an absolute position from the <code>positions</code> array. |

Glyph Shapes

<code>tangents</code>	An array of tangents that determines the scaling or rotation of each glyph. If you pass <code>nil</code> , <code>GXDrawGlyphs</code> does not apply optional scaling or rotation to each character.
<code>styleRuns</code>	An array of values that specify how many consecutive glyphs use the same style; the styles are stored in the style list. The number of entries in the style-runs array must match the number of entries in the <code>glyphStyles</code> parameter; the sum of the entries in the style-runs array should equal the value in the <code>charCount</code> parameter, so that every glyph is part of a style run. The value of each entry must be greater than or equal to 1. If you pass <code>nil</code> for this parameter, <code>GXDrawGlyphs</code> uses the default style object when drawing the glyphs.
<code>glyphStyles</code>	An array of references to style objects. This array should be present only if the style-runs array is present. If you pass <code>nil</code> , <code>GXDrawGlyphs</code> assigns the glyphs the default style.

DESCRIPTION

The `GXDrawGlyphs` function draws the text using the default transform and ink objects. If the `glyphStyles` parameter contains no values, `GXDrawGlyphs` also uses the default style object. The function does not create a new glyph shape that you can subsequently use.

The parameter `charCount` indicates the number of characters present, which may not necessarily equal the number of bytes in the length of the `text` parameter. The interpretation of characters depends on the glyph shape's style attribute.

ERRORS, WARNINGS, AND NOTICES

Errors

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

SEE ALSO

To draw an existing glyph shape, use the `GXDrawShape` function, described in the “Shape Objects” chapter in *Inside Macintosh: QuickDraw GX Objects*.

To draw simple text that shares the same font and other font characteristics, see the description of the `GXDrawText` function in the chapter “Text Shapes” in this book.

To draw a layout shape, see the description of the `GXDrawLayout` function in the chapter “Layout Shapes” in this book.

Getting and Setting the Properties of Glyph Shapes

You can get the settings of the various arrays of glyph shapes—the positions, advance bits, tangents, style-runs arrays, and the styles—with the `GXGetGlyphs` function. You can change any type of shape into a glyph shape with the `GXSetGlyphs` function.

Glyph Shapes

The `GXGetGlyphParts` function allows you to examine parts of a glyph shape, such as some of the character codes stored in the shape or a portion of the positions array. The `GXSetGlyphParts` function allows you to change parts of a glyph shape.

The `GXGetGlyphPositions` function allows you to examine the positions and advance bits arrays of a glyph shape. The `GXSetGlyphPositions` function lets you change the values of the positions and advance bits arrays.

The `GXGetGlyphTangents` function allows you to examine the tangents array of a glyph shape. The `GXSetGlyphTangents` function lets you change the values of the tangents array.

GXGetGlyphs

You can use the `GXGetGlyphs` function to retrieve the information in the geometry of a glyph shape.

```
long GXGetGlyphs(gxShape source, long *charCount,
                 unsigned char text[], gxPoint positions[],
                 long advance[], gxPoint tangents[],
                 long *runCount, short styleRuns[],
                 gxStyle glyphStyles[]);
```

<code>source</code>	A reference to the glyph shape.
<code>charCount</code>	A pointer to a long value. On return, the value is the number of character codes in the <code>text</code> parameter. For non-Roman scripts, the byte length may be up to twice the number of character codes.
<code>text</code>	An array of character codes or glyph codes. On return, the array contains the character codes of the source glyph shape.
<code>positions</code>	An array of points. On return, the array contains the positions array of the source glyph shape.
<code>advance</code>	An array of bits. On return, the array contains the advance bits array of the source glyph shape.
<code>tangents</code>	An array of points that serve as tangents. On return, the array contains the tangents array of the source glyph shape.
<code>runCount</code>	A pointer to a long value. On return, the value is the number of style runs.
<code>styleRuns</code>	An array of short values. On return, each element specifies how many consecutive glyphs use the same style (the style runs array).
<code>glyphStyles</code>	An array of references to style objects. On return, the array contains the style list for the source glyph shape.

function result The number of bytes needed to store the character codes of the glyph shape.

Glyph Shapes

DESCRIPTION

The `GXGetGlyphs` function gets information from a glyph shape: the character or glyph codes, positions, advance bits, tangents, style runs, and style lists. The parameters, except for the shape reference, are pointers to storage for the other information, such as the number of characters (which may not be equal to the number of bytes, depending on the platform) or the positions array of the shape.

You must allocate the required space for the `charCount`, `text`, `positions`, `advance`, `tangents`, `styleRuns`, and `glyphStyles` parameters.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`shape_is_nil`

`illegal_type_for_shape`

(debugging version)

Notices (debugging version)

`glyph_tangents_have_no_effect`

`glyph_positions_determined_by_advance`

SEE ALSO

To retrieve the description of a text shape, use the `GXGetText` function, described in the chapter “Text Shapes” in this book.

GXSetGlyphs

You can use the `GXSetGlyphs` function to change the information in a glyph shape.

```
void GXSetGlyphs(gxShape target, long charCount,
                 const unsigned char text[],
                 const gxPoint positions[], const long advance[],
                 const gxPoint tangents[], const short styleRuns[],
                 const gxStyle glyphStyles[]);
```

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

charCount The number of character codes in the `text` parameter. For non-Roman scripts, the byte length may be up to twice the number of character codes. This value must be greater than or equal to 0; if it is not 0, the `text` parameter must point to an array of character codes.

text An array of character codes or glyph codes. Whether each character code is 8-bit or 16-bit depends on both the platform in the glyph shape’s style object and on the actual byte values. If the value of the shape attribute `gxIgnorePlatformShape` is clear, then it returns an array of character codes. If it is set, it returns an array of 16-bit glyph codes.

Glyph Shapes

<code>positions</code>	An array of points, one for each character code in the <code>text</code> parameter. Depending on the corresponding values in the <code>advance</code> bits array, these points are either the relative or absolute positions of the glyphs in geometry space. If you pass <code>nil</code> for this parameter, <code>GXSetGlyphs</code> places the first glyph at (0.0,0.0).
<code>advance</code>	An array of bits, one bit for each character code in the <code>text</code> parameter. The high bit in the long integer corresponds to the first character code in the shape. If the bit is set, then the corresponding position for that glyph in the <code>positions</code> array is an absolute position in geometry space. If the bit is not set, the position for that glyph is relative to the end of the advance width of the previous glyph. If you pass <code>nil</code> for both this parameter and the <code>positions</code> parameter, <code>GXSetGlyphs</code> positions each glyph relative to the advance width of the preceding glyph. If you pass <code>nil</code> for this parameter but not for the <code>positions</code> parameter, <code>GXSetGlyphs</code> gives each glyph an absolute position from the <code>positions</code> array.
<code>tangents</code>	An array of points that serve as tangents. This array determines the scaling or rotation of each glyph. If you pass <code>nil</code> , <code>GXSetGlyphs</code> does not apply optional scaling or rotation to each character.
<code>styleRuns</code>	An array of values that specify how many consecutive glyphs use the same style; the styles are stored in the style list. The sum of the entries in the style runs array should equal the value in the <code>charCount</code> parameter. The value of each entry must be greater than or equal to 1. If you pass <code>nil</code> for this parameter, <code>GXSetGlyphs</code> uses the default style object for all glyphs in the shape.
<code>glyphStyles</code>	An array of references to style objects. This array should be present only if the value of the <code>styleRuns</code> parameter is not <code>nil</code> . If you pass <code>nil</code> for <code>glyphStyles</code> , <code>GXSetGlyphs</code> assigns the style object associated with the shape object to all glyphs in the shape. If an entry in the style list is <code>nil</code> , that entry refers to the style object associated with the shape.

DESCRIPTION

The `GXSetGlyphs` function changes the information in a glyph shape. If you use `GXSetGlyphs` to change one of the elements of one of the arrays in a glyph shape—for example, a point in the `positions` array—you must send the entire array for the shape.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>missing_parameter</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>cannot_add_unspecified_new_glyphs</code>	(debugging version)
<code>style_run_array_does_not_match_number_of_characters</code>	(debugging version)

Glyph Shapes

Warnings

<code>shape_access_not_allowed</code>	(debugging version)
<code>first_glyph_advance_must_be_absolute</code>	(debugging version)

SEE ALSO

To change a portion of an array in a glyph shape, use the `GXSetGlyphParts` function, described on page 4-30.

GXGetGlyphParts

You can use the `GXGetGlyphParts` function to retrieve parts of a glyph shape.

```
long GXGetGlyphParts(gxShape source, long index, long charCount,
                    long *byteLength, unsigned char text[],
                    gxPoint positions[], long advanceBits[],
                    gxPoint tangents[], long *runCount,
                    short styleRuns[], gxStyle styles[]);
```

<code>source</code>	A reference to the shape whose information you want to retrieve.
<code>index</code>	The number of the first character you want to retrieve.
<code>charCount</code>	The number of character codes you want to retrieve. The number of actual glyphs returned may be different from this value. (The function itself returns the number of glyphs retrieved.)
<code>byteLength</code>	A pointer to a long value. On return, the value is the number of bytes used by the character codes.
<code>text</code>	An array of character codes. On return, the array contains the character codes from the source shape.
<code>positions</code>	An array of <code>gxPoint</code> values. On return, the array contains the positions of each of the glyphs.
<code>advanceBits</code>	An array of long values. On return, the array contains the advance bits for the glyph shape.
<code>tangents</code>	An array of <code>gxPoint</code> values. On return, the array contains the tangents for glyphs you specify.
<code>runCount</code>	A pointer to a long value. On return, the value is the number of style runs.
<code>styleRuns</code>	An array of short values. On return, the array contains the number of glyphs associated with each style run.
<code>styles</code>	An array of style-object references. On return, the array contains the styles of the glyphs you specify.

Glyph Shapes

function result The number of glyphs retrieved.

DESCRIPTION

The `GXGetGlyphParts` function retrieves the specified glyphs from the source shape, which must be of type `glyphsType`, and places the character codes into the `text` parameter, the positions into the `positions` parameter, the advance bits into the `advanceBits` parameter, and so on. It also returns the length in bytes of these characters in the `byteLength` parameter and the number of style runs in the `runCount` parameter. You must allocate the memory to store the information the function returns.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)

Warnings

<code>index_out_of_range</code>
<code>count_out_of_range</code>

SEE ALSO

To retrieve part of a text shape, use the `GXGetTextParts` function, described in the chapter “Text Shapes” in this book.

GXSetGlyphParts

You can use the `GXSetGlyphParts` function to replace specific parts of a glyph shape with new information.

```
void GXSetGlyphParts(gxShape source, long index,
                    long oldCharCount, long newCharCount,
                    const unsigned char text[],
                    const gxPoint positions[],
                    const long advanceBits[],
                    const gxPoint tangents[],
                    const short styleRuns[],
                    const gxStyle styles[]);
```

<code>source</code>	A reference to the shape you want to change.
---------------------	--

Glyph Shapes

<code>index</code>	The index number of the first character code that you want to replace in the source shape. This value must be greater than or equal to 1.
<code>oldCharCount</code>	The number of character codes that you want to replace in the source shape. This value may be greater than or equal to 0, or it may be <code>gxSelectToEnd</code> , which selects all characters after and including the character code specified by the value of the <code>index</code> parameter.
<code>newCharCount</code>	The number of character codes you want to add to the source shape.
<code>text</code>	An array of the new characters you want to add.
<code>positions</code>	The new values for the positions array.
<code>advanceBits</code>	The new values for the advance bits array.
<code>tangents</code>	The new values for the tangents array.
<code>styleRuns</code>	The new style-runs array.
<code>styles</code>	The new style list.

DESCRIPTION

The `GXSetGlyphParts` function replaces the specified glyphs in the source shape with the new glyphs specified by the `text`, `positions`, `advanceBits`, `tangents`, `styleRuns`, and `styles` parameters.

If you send new text to the shape, the values in the `positions`, `advanceBits`, `tangents`, `styleRuns`, and `styles` parameters apply only to new text, not to the shape as a whole.

You cannot change the style list and style-runs array independently of each other. If you change one, you must resend the values for the other.

Using the `GXSetGlyphParts` function, you can change the positions and advance bits arrays independently, and you can send part of or all of a positions or advance bits array.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>inconsistent_parameters</code>	(debugging version)
<code>index_is_less_than_zero</code>	(debugging version)
<code>count_is_less_than_zero</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)
<code>cannot_add_unspecified_new_glyphs</code>	(debugging version)
<code>style_run_array_does_not_match_number_of_characters</code>	(debugging version)

Warnings

<code>index_out_of_range</code>

Glyph Shapes

Errors

<code>count_out_of_range</code>	
<code>shape_access_not_allowed</code>	(debugging version)
<code>first_glyph_advance_must_be_absolute</code>	(debugging version)

SEE ALSO

For more information on how to use the `GXSetGlyphParts` function, see “Changing Parts of a Glyph Shape” beginning on page 4-13.

To change parts of a text shape, use the `GXSetTextParts` function, described in the chapter “Text Shapes” in this book.

GXGetGlyphPositions

You can use the `GXGetGlyphPositions` function to retrieve a number of entries in the positions and advance bits arrays of a glyph shape.

```
long GXGetGlyphPositions(gxShape source, long index, long count,
                        long advance[], gxPoint positions[]);
```

<code>source</code>	A reference to the glyph shape whose positions and advance bits arrays you want to retrieve.
<code>index</code>	The first entry in the positions and advance bits arrays you want.
<code>count</code>	The number of entries in the positions and advance bits arrays to return, starting at the value of the <code>index</code> parameter. This value may be greater than or equal to 0. If you pass 0, the function result is a Boolean value indicating whether the glyph shape has any entries set in the advance bits array. A function result of <code>false</code> indicates that all the advance bits array values are the default values.
<code>advance</code>	An array of long values. On return, the array contains the requested portion of the advance bits.
<code>positions</code>	An array of points. On return, the array contains the requested portion of the positions.
<i>function result</i>	The number of retrieved entries, if the value of <code>count</code> is greater than 0; otherwise, a Boolean value indicating whether the glyph shape has any entries set in the advance bits array.

DESCRIPTION

The `GXGetGlyphPositions` function retrieves the specified number of entries from the positions and advance bits arrays of the source glyph shape.

If the glyph shape’s advance bits array is not set, `GXGetGlyphPositions` fills the advance bits array with zeros. If the glyph shape’s positions array is empty, the function

Glyph Shapes

returns the positions array filled with the coordinate point (0.0,0.0), unless it is also returning the first point in the shape; in this case, the function returns the initial position of the glyph shape and fills the rest of the array with (0.0,0.0).

If you do not want either the advance bits or positions array, send `nil` for that parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_zero</code>	(debugging version)
<code>illegal_type_for_shape</code>	(debugging version)
<code>out_of_memory</code>	

Warnings

<code>index_out_of_range</code>
<code>count_out_of_range</code>

GXSetGlyphPositions

You can use the `GXSetGlyphPositions` function to replace the selected positions and advance bits arrays of a glyph shape with new arrays.

```
void GXSetGlyphPositions(gxShape target, long index, long count,
                        const long advance[],
                        const gxPoint positions[]);
```

<code>target</code>	A reference to the shape whose advance bits and positions arrays you want to replace.
<code>index</code>	The first entry in the positions and advance bits arrays you want to replace.
<code>count</code>	The number of entries to replace. This value may be greater than or equal to 1. It may also be equal to <code>gxSelectToEnd</code> , which indicates that <code>GXSetGlyphPositions</code> should select all positions and advance bits beginning at the entry indicated by <code>index</code> .
<code>advance</code>	The advance bits array you want to put into the target shape. If you pass <code>gxSetToNil</code> , <code>GXSetGlyphPositions</code> sets all the bits, except the first one, in this array to 0.
<code>positions</code>	The positions array you want to put into the target shape. If you pass <code>gxSetToNil</code> , <code>GXSetGlyphPositions</code> sets all the positions, except the first one, in this array to (0.0,0.0).

Glyph Shapes

DESCRIPTION

The `GXSetGlyphPositions` function replaces the selected sections of the advance bits and positions arrays of the target shape with the arrays in the `advance` and `positions` parameters.

ERRORS, WARNINGS, AND NOTICE

Errors

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

Warnings

<code>index_out_of_range</code>	
<code>count_out_of_range</code>	
<code>shape_access_not_allowed</code>	
<code>first_glyph_advance_must_be_absolute</code>	(debugging version)

Notices (debugging version)

<code>glyph_positions_are_already_set</code>	
--	--

GXGetGlyphTangents

You can use the `GXGetGlyphTangents` function to get some or all of the information about tangents in a glyph shape.

```
long GXGetGlyphTangents(gxShape source, long index, long count,
                        gxPoint tangents[]);
```

<code>source</code>	A reference to the shape from which you want the tangent information.
<code>index</code>	The index number of the first tangent to return. This value must be greater than or equal to 1.
<code>count</code>	The number of tangents to return. This value may be greater than or equal to 0. If you pass 0, the function result changes to a Boolean value indicating whether the glyph shape has any explicit tangents at all.
<code>tangents</code>	An array of <code>gxPoint</code> tangents. On return, the array of tangents.

<i>function result</i>	The number of retrieved tangents, or, if the value of <code>count</code> is 0, a Boolean value indicating whether the glyph shape has any explicit tangents. A function result of <code>false</code> indicates that all the tangents are the default settings.
------------------------	--

DESCRIPTION

The `GXGetGlyphTangents` function retrieves the specified number of tangent points from the glyph shape. If the glyph shape does not have tangents, then `GXGetGlyphTangents` returns an array filled with the value (1.0,0.0) in the `tangents` parameter.

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)
 - `illegal_type_for_shape` (debugging version)
 - `parameter_out_of_range` (debugging version)
- Warnings**
 - `index_out_of_range`
 - `count_out_of_range`

GXSetGlyphTangents

You can use the `GXSetGlyphTangents` function to replace selected tangents in a glyph shape with new ones.

```
void GXSetGlyphTangents(gxShape target, long index, long count,
                        const gxPoint tangents[]);
```

- | | |
|-----------------------|--|
| <code>target</code> | A reference to the glyph shape. |
| <code>index</code> | The first tangent you want to replace. |
| <code>count</code> | The number of tangents to replace. This value may be greater than or equal to 1. It may also be equal to <code>gxSelectToEnd</code> , which indicates that the function should select all tangents, from the one indicated by <code>index</code> to the end of the tangents array. |
| <code>tangents</code> | The array of tangents you want to put into the glyph shape. You can pass <code>gxSetToNil</code> for an entry in this array; this value sets the tangents to the default value of (1.0,0.0). |

DESCRIPTION

The `GXSetGlyphTangents` function replaces the selected tangents with the new ones you pass in the `tangents` parameter.

Glyph Shapes

ERRORS, WARNINGS, AND NOTICES

Errors

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

Warnings

<code>index_out_of_range</code>
<code>count_out_of_range</code>
<code>shape_access_not_allowed</code>

Notices (debugging version)

<code>glyph_tangents_already_set</code>

Summary of Glyph Shapes

Functions

Creating and Drawing Glyph Shapes

```

gxShape GXNewGlyphs      (long charCount, const unsigned char text[],
                          const gxPoint positions[],
                          const long advance[],
                          const gxPoint tangents[],
                          const short styleRuns[],
                          const gxStyle glyphStyles[]);

void GXDrawGlyphs        (long charCount, const unsigned char text[],
                          const gxPoint positions[],
                          const long advance[], const gxPoint tangents[],
                          const short styleRuns[],
                          const gxStyle glyphStyles[]);

```

Getting and Setting the Properties of Glyph Shapes

```

long GXGetGlyphs          (gxShape source, long *charCount,
                          unsigned char text[], gxPoint positions[],
                          long advance[], gxPoint tangents[],
                          long *runCount, short styleRuns[],
                          gxStyle glyphStyles[]);

void GXSetGlyphs          (gxShape target, long charCount,
                          const unsigned char text[],
                          const gxPoint positions[],
                          const long advance[],
                          const gxPoint tangents[],
                          const short styleRuns[],
                          const gxStyle glyphStyles[]);

long GXGetGlyphParts      (gxShape source, long index, long charCount,
                          long *byteLength, unsigned char text[],
                          gxPoint positions[], long advanceBits[],
                          gxPoint tangents[], long *runCount,
                          short styleRuns[], gxStyle styles[]);

void GXSetGlyphParts      (gxShape source, long index, long oldCharCount,
                          long newCharCount, const unsigned char text[],
                          const gxPoint positions[],
                          const long advanceBits[],
                          const gxPoint tangents[],
                          const short styleRuns[],
                          const gxStyle styles[]);

```

Glyph Shapes

```
long GXGetGlyphPositions (gxShape source, long index, long count,  
                          long advance[], gxPoint positions[]);  
void GXSetGlyphPositions (gxShape target, long index, long count,  
                          const long advance[],  
                          const gxPoint positions[]);  
long GXGetGlyphTangents (gxShape source, long index, long count,  
                        gxPoint tangents[]);  
void GXSetGlyphTangents (gxShape target, long index, long count,  
                        const gxPoint tangents[]);
```