

Typographic Styles

This chapter describes how typographic shapes use the style object. Read this chapter if you want to create or use any styles with QuickDraw GX typographic shapes.

Before reading this chapter, you should be familiar with the information in the chapter “Introduction to QuickDraw GX Typography” in this book. You should also be familiar with typographic shapes, as discussed in the chapter “Typographic Shapes” in this book.

For more information on style objects, see the chapter “Style Objects” in *Inside Macintosh: QuickDraw GX Objects*.

This chapter does *not* cover style object properties used exclusively by layout shapes, such as run controls and the kerning adjustments array, which are also part of the style object. For more information about style properties used by layout shapes, see the chapter “Layout Styles” in this book.

This chapter introduces the properties of the style object and how they are associated with QuickDraw GX typographic shapes. It then shows how to use QuickDraw GX functions to

- get and set the style object’s properties, such as its font, text face, text size, and text attributes
- create a text face in order to create stylistic variations of a font
- orient text vertically instead of horizontally

About Typographic Styles

Typographic styles exist to provide information about typographic shapes. Each QuickDraw GX typographic style associated with a particular typographic shape defines much of that shape’s appearance, such as what font the shape uses and where it is placed. Typographic styles also describe what glyphs are represented by a font’s character encoding and its text size, as well as the stylistic variations of a font and its text faces.

Typographic styles are device-independent. The styles of a typographic shape are not affected by the properties of the display device for which the shape is drawn.

Although a typographic style is the same as a geometric style, typographic shapes—text, glyph, and layout shapes—use different properties from the style object than those used by geometric shapes, such as caps, joins, and dashes.


Style Properties Associated With Typographic Shapes

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

Typographic Styles

Style objects have 22 accessible properties, as shown in Figure 6-1. Note that, because a style is an object and not a data structure, the order of the properties as shown in Figure 6-1 is completely arbitrary.

Figure 6-1 The style object used by all typographic shapes

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

Thirteen of the style object's properties pertain only to typographic styles—that is, styles associated with typographic shapes. Seven apply to all typographic shapes:

- **Font.** A reference to the font to use in drawing the text of this shape.
- **Text face.** The text face—the constructed stylistic variation from plain text—to apply in drawing the text of this shape.
- **Text size.** The size, in typographic points (72 per inch), at which to draw the text of this shape.
- **Alignment.** The alignment value to use when drawing the text of this shape. Text may be left-aligned, right-aligned, anywhere between the two alignments (such as centered), or fully justified.
- **Font variations.** The list of font variations—stylistic variations built into the font—available for drawing the text of this shape.
- **Encoding.** The type of character encoding used to represent the text of this shape, as well as its script and language.
- **Text attributes.** The set of flags that allow you to specify how QuickDraw GX alters glyph outlines or chooses the proper metrics for horizontal or vertical text.

Most of the properties of the style object associated with geometric shapes—pen size, cap, join, dash, curve error, and style attributes—can be set in a style object used by a typographic shape, but they do *not* affect the appearance of the shape. (Patterns are an exception; they do affect the appearance of a typographic shape. See Figure 6-20

Typographic Styles

on page 6-33 for example.) However, you can take advantage of the geometric style properties by using the text face property of the style object. In this way, you can apply cap, join, dash, and pattern properties to your typographic shape. See the section “Applying Patterns and Dashes to Text Faces” on page 6-32.

Note

Both glyph shapes and layout shapes may have arrays of styles in their shape geometry and therefore do not necessarily use the style object associated with the shape object. However, any operation you can perform on the shape’s style object can also be performed on the styles stored in the shape’s geometry. (Note that any `nil` member of an array causes the shape’s style object to be used.) ♦

QuickDraw GX provides functions for manipulating each of these style object properties. The properties and functions pertaining to typographic styles are described in the following sections.

Font

The font property of style objects specifies the font or font family of a style object. QuickDraw GX provides your application with functions for retrieving and specifying font information for a style object, as well as retrieving and specifying information for a style object associated with a particular shape. The functions for getting and setting the font of a style object are described in the “Typographic Styles Reference” section of this chapter.

For more information about fonts—specifically, encodings and font variations—see the chapter “Font Objects” in this book.

Text Face

If your application needs to apply a typestyle to text, you have three options. (A **typestyle** is a specific variation in the appearance of a glyph that can be applied consistently to all the glyphs in a font family.) You can

- use a font specifically designed for that typestyle, as described in the chapter “Font Objects”
- use a font that has a font variation for that typestyle, as described in the chapter “Font Objects”
- apply a text face created by your application to that text

The first two methods involve using information already present in the font. If there isn’t a separate font in the appropriate typestyle, or the font doesn’t contain the font variation that the user needs, your application can use text faces to create the desired typestyle. A **text face** is a typestyle created by an algorithmic method, which allows you to control the appearance and placement of glyphs in a font.

A text face consists of a mapping, which affects the advance widths and interglyph spacing—but not the shape of the glyphs—and a number of optional face layers, which describe the appearance of the glyphs.

Typographic Styles

A **face layer** specifies the manner in which the glyphs are drawn. When all of the face layers are combined, they form the visual composite. This creates the particular look of the text face—that is, bold, oblique, underline, condensed, extended, or a more unusual pattern, dash, join, or other style. If a text face has multiple face layers, the layers combine to form the final text face, as shown in Figure 6-2.

A face layer contains the following elements:

- a shape fill
- an optional style object
- an optional transform object
- bold values for the x and y directions
- layer flags

The Shape Fill

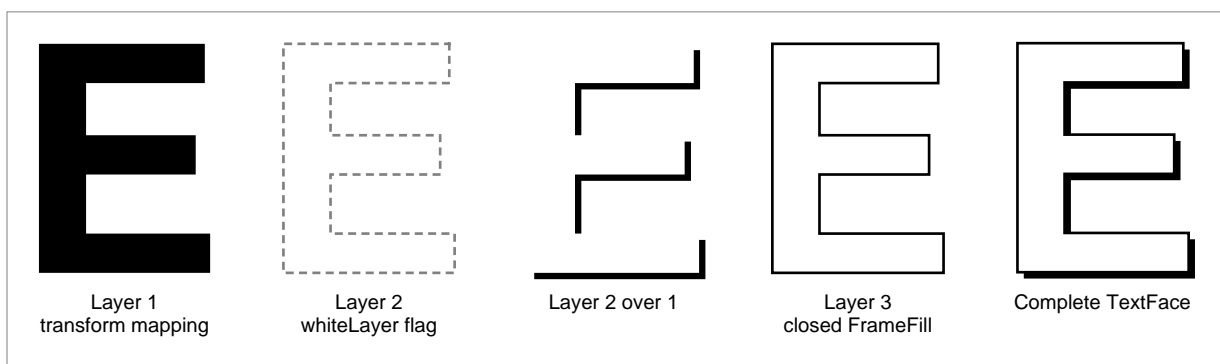
The shape fill determines the kind of geometry—that is, fill or frame—used by the layer. If the fill is an open frame or closed frame fill, then the glyph is converted to a path shape.

For more information about shape fills, see the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

The Style

The style in a face layer allows the text face to take advantage of the geometric, as opposed to typographic, properties of the style object. The only restriction is that the text face’s style cannot have its own text face.

Figure 6-2 Face layers combined to form the visual composite of a Roman “E”



Typographic Styles

The style is the overriding style for the text face. The values in this style are interpreted in a unique way because of the text face. The style's pen size, pattern, dash, and join are scaled by a factor corresponding to the point size of the text.

The fields of the outline style are multiplied by the text size of the text being drawn. Thus, a pattern in an outline style is scaled relative to the text being patterned, so that, for example, a 30.0-point "A" will have a pattern shape twice as large as a 15.0-point "A".

The bulleted items listed are the fields of the outline style that are scaled. Wherever the text "scaled by textSize" appears, the text size refers to the size of the text being drawn. This text size comes from the text size field of the text's style. The scaled fields include:

- Pen width: scaled by the text size
- Dash (if present): advance scaled by the text size; dash shape scaled in X only by the text size
- Pattern (if present): u and v vectors scaled in X and Y by the text size; pattern shape scaled in X and Y by the text size
- Start and End Caps (if present): scaled in X and Y by the text size.
- Join (if present and join type is `gxShapeJoin`): join shape is scaled in X and Y by the text size.
- Text Size: scaled by the text size

If the outline style's text size is 1.1, for example, the text will end up being drawn with a point size 10 percent larger than if there were no text face. A 12.0 point text will be drawn at 13.2 points.

You can use the style in the text face to pattern a typographic shape. See "Applying Patterns and Dashes to Text Faces" on page 6-32.

The Transform

The transform in the face layer allows you to produce oblique, condensed, and extended typestyles, as well as unusual special effects. You can change, skew, and scale the text face's transform as you would any other transform.

For more information about the transform object, its properties, and the `GXNewTransform`, `GXSkewTransform`, `GXScaleTransform`, and `GXSetTransformMapping` functions, see *Inside Macintosh: QuickDraw GX Objects*.

The Bold Outset

The x and y values in the `boldOutset` field of the face layer scale the thicknesses of the glyphs in the displayed shape. The values for bold are treated as though they are for a 1-point font. For example, to get 3 points worth of bold at 48-point text, you set the bold outset to `ff(3)/ff(48)`.

Layer Flags

The **layer flags** describe the characteristics of one layer of a text face. They are used primarily to determine the underlining capabilities of the text face. By setting the layer flags, you can affect where the underlining goes. Figure 6-3 indicates underlining a glyph with tangent values.

Figure 6-3 An underlined glyph with tangent values



QuickDraw GX allows your application to

- underline all of the glyphs in that text face
- underline only the glyphs that make up words (skipping whitespace glyphs)
- connect text of different text faces

Figure 6-4 shows underlining with intervals, and with a style change from italic to Roman.

Figure 6-4 Underlining with interval and with style changes

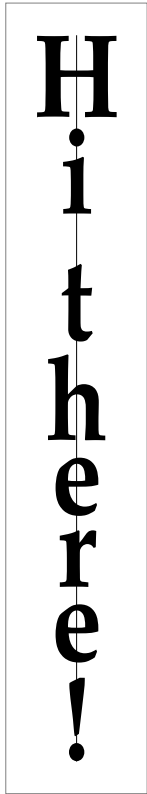


Typographic Styles

A layer flag can also indicate whether the layer adds to or subtracts from the previous layers.

Figure 6-5 shows underlining vertical text, with centering.

Figure 6-5 Underlining vertical text through its center



Typographic Styles

Table 6-1 describes values for layer flags. When you pass a value for one of the flags, QuickDraw GX performs the described action.

Table 6-1 Layer flag values and descriptions

Flag	Value	Description
<code>gxUnderlineAdvanceLayer</code>	1	Draws an underline from the beginning of the text that shares one text face to the end, including white spaces.
<code>gxSkipWhiteSpaceLayer</code>	2	Does not draw a line under glyphs that have no contours (such as the space character) if the <code>gxUnderlineAdvanceLayer</code> bit is also set.
<code>gxUnderlineIntervalLayer</code>	4	Extends the underline through the gaps between text. If you set this bit, you must also set the <code>gxStringLayer</code> bit.
<code>gxUnderlineContinuationLayer</code>	8	Draws an underline across text of different style runs. If you set this bit, you must also set the <code>gxStringLayer</code> bit. Also, you must set the <code>gxUnderlineAdvanceLayer</code> flag; if you do not, you get an error.
<code>gxWhiteLayer</code>	16	Subtracts portions of previously drawn layers. You can set this bit only for the second or greater layer.
<code>gxClipLayer</code>	32	Clips the underline with the outline as a glyph.
<code>gxStringLayer</code>	64	Connects the text of different text faces and style runs.

Text Size

The text size property of style objects specifies the size, in fractional typographic points, of the text of a style object.

QuickDraw GX provides functions to retrieve and specify the text size from a style object as well as functions to retrieve and specify the text size for the style object associated with a particular shape. The functions for getting and setting the text size of a style object are described later in the “Typographic Styles Reference” section of this chapter.

Alignment

Alignment is the process of placing text in relation to one or both margins, which are the left and right sides (or top and bottom sides) of the text area. Text can be aligned left, right, center, or at full justification (full justification allows you to “stretch” or “shrink” a line of text to fit within a given width). Alignment should be used only to compensate for the differences between ideal measurements of characters and device-specific measurements of characters.

The alignment is set by getting and setting the alignment values of the style object. Table 6-2 describes some alignment values of the style object.

Table 6-2 Alignment values and descriptions

Alignment	Value	Description
Left	0.0	Text is drawn to the right of the left margins (or below the top margin, for vertical text).
Right	1.0	Text is drawn to the left of the right margin (or above the bottom margin, for vertical text).
Center	0.5	Text is drawn between the left and right margins with an equal amount of space on either side.
Full Justification	-1.0	Text is evenly distributed between the margins (left and right for horizontal text or top and bottom for vertical text) because the white space on the line is distributed between the words and, to a lesser extent, between the glyphs in the words.

Note

The alignment value in the style object is used only by text and glyph shapes. The layout shape has its own method of justifying and aligning text, as described in the chapter “Layout Line Control.” ♦

As shown in Table 6-2, several values are predefined for different types of alignment. However, any fractional value between 0.0 and 1.0 is legal for alignment and represents a gradation of one of the types of alignment. (Note that values between 0.0 and -1.0 are illegal.) For example, a value of 0.25 indicates that QuickDraw GX should draw the shape one quarter of the way from the left text position, or one quarter of the way from the left and right edges.

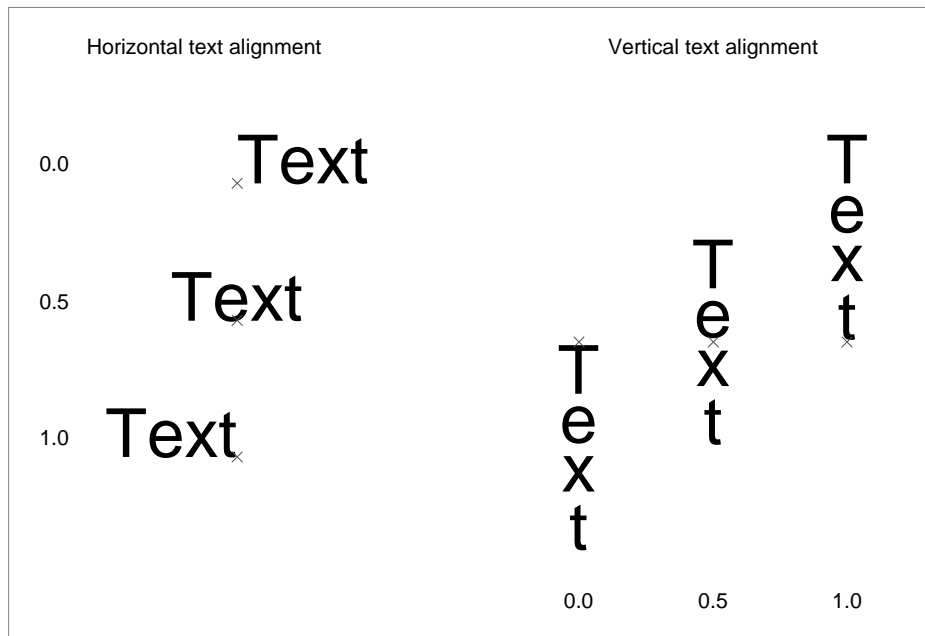
Typographic Styles

Figure 6-6 compares alignment values for horizontal and vertical text.

IMPORTANT

If you use text and glyph shapes with non-Roman scripts, you are not assured of getting the proper results when you use these alignment values. For example, if you use a script such as Arabic that relies on kashidas (extending bars), then full justification may break that alignment apart. To guarantee correct results in non-Roman cases, you should use layout shapes. ▲

Figure 6-6 Comparing alignment values for horizontal and vertical text

**Full Justification**

In full justification, the glyphs are distributed evenly between the margins, the left and right margins for horizontal text or the top and bottom margins for vertical text. Full justification does not affect text or layout shapes; it affects only glyph shapes that have two absolute positions. The first absolute position is first entry in the glyph shape's position array; the second absolute position is the final entry in the glyph shape's position array.

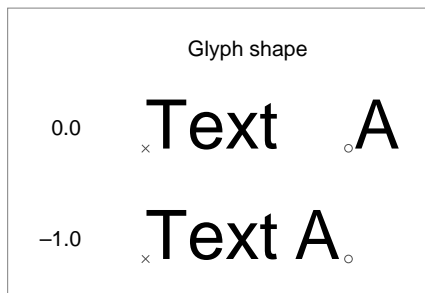
For each pair of absolute positions in the glyph shape, the first position specifies the left alignment for the glyph that corresponds to that position; the second of the pair specifies the right alignment for the glyph that corresponds to that position. QuickDraw GX justifies the remaining glyphs between the pair of glyphs corresponding to the absolute positions. The glyphs are justified using the glyph advance and the relative position.

Typographic Styles

QuickDraw GX justifies the whitespace glyphs (whitespace glyphs are those with no contours, such as spaces between characters) after justifying the other glyphs. If there is a difference between the advance after the next-to-the-last glyph and the initial position of the last glyph, the difference is divided by the number of whitespace glyphs. If there are no whitespace glyphs, or if changing their width is not enough to compensate for the mismatch in widths, the positions of all other characters are also adjusted.

Figure 6-7 shows the absolute positions for two different alignment values.

Figure 6-7 Comparing alignment values for full justification



Font Variations

The font variations property of a style object specifies the font variations that will be applied to the style's font.

QuickDraw GX provides your application with functions to retrieve and specify the font variations from a style object, as well as to retrieve and specify the font variations associated with a particular shape. See "Getting and Setting the Font Variations of a Style Object" on page 6-51 and the chapter "Font Objects" in this book.

The QuickDraw GX Font Variation Suite

A typographic style may specify the values for any number of variation axes. (A variation axis has a name that identifies the typestyle which the axis represents, a set of maximum and minimum values for the axis, and the default value of the axis.)

A style's font might not support, however, all of the specified axes, or support axes not explicitly mentioned in the style. In addition, the style may specify a value for an axis that is beyond the supported range for that axis by the style's font. QuickDraw GX manages all of these cases by converting the style's list of variations into a canonical form before using them. This canonical form is called a **font variation suite**.

The canonical form is a complete listing of every axis supported in the font (see the `GXCountFontVariations` function described in the "Font Objects" chapter in this book), in the order specified by the font (see the `GXGetFontVariation` function). Each axis is given a value. If the style specifies a value for an axis, the value is pinned to lie within (inclusively) the font's minimum and maximum value for that axis. Axes not specified in the style are set to their default values.

Typographic Styles

The functions `GXGetStyleFontVariationSuite` and `GXGetShapeFontVariationSuite` return a `gxFontVariation` array in this canonical form. For more information, see “Retrieving the Elements in a Font Variation Suite” on page 6-55.

The functions return the number of elements in the variation suite for the font specified by the style or shape. This number is the same as the number of font variation axes returned by `GXCountFontVariations`, described in the chapter “Font Objects” in this book. If the variations parameter is not `nil`, the variations specified in the style are converted into their canonical form, and then copied into variations.

Font Metrics

The font metrics property allows you to retrieve font metrics for specified style objects or style objects associated with a shape object.

You can retrieve the font metrics, including line spacing and caret angle, for a style object. You can also retrieve the font metrics for the style object associated with a shape object. You can retrieve the font metrics for the style object associated with a shape, taking into account the shape’s transform or the mappings on the specified view port and view device. See “Retrieving Font Metrics” on page 6-57.

Encoding

The encoding property of a style object represents a combination of its platform, script, and language. Platforms, scripts, and languages are described in the chapter “Font Objects” in this book.

QuickDraw GX provides your application with functions to retrieve and specify the encoding information of a style object, as well as to retrieve and specify the encoding information for the style object associated with a particular shape. The functions are described in “Getting and Setting the Encoding of a Style Object” on page 6-61.

For more information on encoding, see the chapters “Typographic Shapes” and “Font Objects” in this book.

Text Attributes

Each style object has a set of text attributes that modify the behavior of the style object associated with typographic shapes. **Text attributes** are a collection of flags which individually affect different properties of typographic styles.

Table 6-3 describes text attributes and their values.

As shown in Table 6-3, the text attribute flags allow you to specify how QuickDraw GX alters glyph outlines or chooses the proper types of metrics for horizontal or vertical text. For example, the `gxAutoAdvanceText` text attribute determines the difference between the advance width of the glyph from the font and the advance width of the same glyph with the text face applied. It then adds this difference to the original advance width.

Table 6-3 Text attributes and their values

Attribute	Value	Description
<code>gxAutoAdvanceText</code>	0x0001	QuickDraw GX automatically changes the advance width of a glyph to match the change of glyph width due to a text face.
<code>gxNoContourGridText</code>	0x0002	QuickDraw GX doesn't use hinted outlines before displaying the text—that is, it instructs the scaler not to use hints when imaging. If you set this bit, you must also set the <code>gxNoMetricsGridText</code> bit.
<code>gxNoMetricsGridText</code>	0x0004	Uses the ideal metrics to space the glyphs in a typographic shape—that is, instructs the scaler not to use hints when measuring the character.
<code>gxAnchorPointsText</code>	0x0008	Includes data for all the outline's points. QuickDraw GX's default action is to return a basic outline, removing points that do not affect the shape, such as single-point contours. If set, and a typographic shape is converted to a path, all points are returned.
<code>gxVerticalText</code>	0x0010	Uses the vertical advance and side bearing metrics. The default values are the horizontal advance and side bearing metrics.
<code>gxNoOpticalScaleText</code>	0x0020	QuickDraw GX automatically specifies an optical scale variation value equal to the style's text size, if the style's font supports optical scaling variations and the style does not already specify a value for this axis. If set, this bit prevents QuickDraw GX from specifying an optical scale variation value.

Given a specific point size and resolution, hinted outlines produce better-looking text for a specific point size and resolution by moving points or otherwise altering the outline. A hinted outline for a particular point size that is converted to a path or polygon shape may not respond to scaling as well as graphic objects or unhinted outlines do on a high-resolution display device. The `gxNoContourGridText` text attribute does not use hinted outlines before displaying the text.

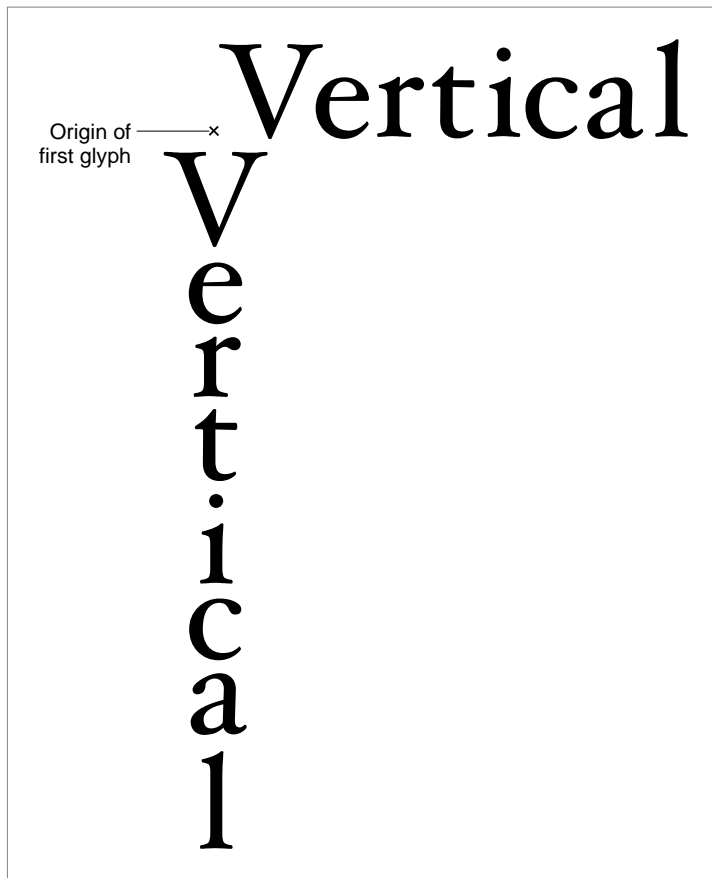
The `gxNoMetricsGridText` text attribute uses the ideal metrics to space the glyphs in a typographic shape. Ideal metrics always give the same interglyph spacing on a high-resolution device, but you may get poorer spacing if you use ideal metrics on a low-resolution device such as the screen. Also, ideal metrics scale linearly with the shape's transform and device. (Your application may want to use ideal metrics for such features as linebreaking.) The alternative, device-specific metrics, change non-linearly with scaling factors in the transform and device.

Typographic Styles

When you convert a glyph to a path, the `gxAnchorPointsText` text attribute includes data for all the points on a glyph's outline if this bit is set. QuickDraw GX's default action is to return a basic outline, removing points that do not affect the shape, such as single-point contours.

The `gxVerticalText` text attribute uses the vertical advance and side bearing metrics. You should set this bit if you want QuickDraw GX to draw text vertically. The default values are the horizontal advance and side bearing metrics. Figure 6-8 shows an example of orienting text both vertically and horizontally.

Figure 6-8 Orienting text vertically and horizontally



Typographic Properties of the Default Style Object

When QuickDraw GX first creates a style object, that object has default characteristics defined by QuickDraw GX. The typographic properties of the default style object have the following values:

- A `nil` font reference, meaning that the default font is used for text. The default font is described in the chapter “Font Objects” in this book.
- A text size of 12.0

Typographic Styles

- A text attributes value of `gxNoAttributes` (0)
- A scale value (within the dash property) of 1.0
- A joins miter value of 1.0
- A value of 0 or `nil` for all other typographic properties

Using Typographic Styles

This section describes the basic method of manipulating the style object for typographic shapes. Unless otherwise noted, these methods apply to all three typographic shapes. For detailed information on using a specific typographic shape—text, glyph, or layout—see the chapter describing that particular shape.

This section describes how you can

- create text faces in your application
- set text attributes and orient the vertical text attribute
- apply patterns and dashes to text faces
- create unusual effects with text faces

For more information on setting font variations and encoding, see the chapter “Font Objects” in this book.

Creating Text Faces

You can include text faces in your application that create the basic typetypes: bold, italic, condensed, extended, outlined, and underlined. You may also want to include unusual text faces for users.

Note that your application must allocate enough memory to store the text face and all of the face layers of that text face. For example, to create a text face with three layers, you can use the following code:

```
gxTextFace*face;
long      numOfLayers = 3;

face = (gxTextFace *)NewPtr(sizeof(gxTextFace) +
                             (numOfLayers - gxAnyNumber)* sizeof(gxFaceLayer));
face->faceLayers = numOfLayers;
```

You can then set the values of the advance mapping and the individual face layers.

Setting the Advance Mapping

The advance mapping in the text face affects the positions of the glyphs using the text face. It does not affect the size of the glyphs themselves or change the positions of the shape as stored in the shape's geometry.

Listing 6-1 is an example of a routine that shows advance mapping. Note that the translation component of the advance mapping is multiplied by the text size before being applied.

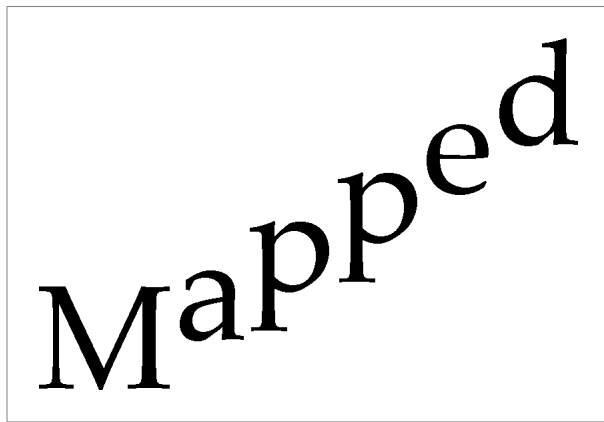
Listing 6-1 Advance mapping

```
static void ShowAdvanceMapping(void)
{
    gxTextFace mappingOnlyFace;
    ResetMapping(&mappingOnlyFace.advanceMapping);
    mappingOnlyFace.advanceMapping.map[2][1] = -fixed1/3;
    mappingOnlyFace.faceLayers = 0;

    GXSetShapeFace(GXGetDefaultShape (gxTextType),
                   &mappingOnlyFace);
    GXDrawText(6, (unsigned char*) "Mapped", nil);
}
```

Figure 6-9 shows the results of executing the code in Listing 6-1.

Figure 6-9 A shape with the advance mapping applied



Note

If you scale the advance mapping, you're scaling on a glyph-by-glyph basis. ♦

Setting a Face Layer

To modify the appearance of the text face, you can use the face layer in a text face. The various layers of the text face are composited on top of one another; the first face layer of the text face is the bottom most layer. Subsequent layers are added to or subtracted from previous layers.

The Transform

To produce italic, condensed, and extended typestyles, as well as unusual effects, you can use the transform in the face layer. You can change, skew, and scale the text face's transform as you would any other transform.

To create an italic text face, for example, you can use the following code in Listing 6-2.

Listing 6-2 Creating an italic text face

```
void ApplyItalicTextFace(gxShape target, Fixed italicValue)
{
    gxTextFace italicFace;

    ResetMapping(&italicFace.advanceMapping);
    italicFace.faceLayers = 1;
    italicFace.faceLayer[0].outlineFill = gxWindingFill;
    italicFace.faceLayer[0].flags = 0;
    italicFace.faceLayer[0].outlineStyle = nil;
    italicFace.faceLayer[0].outlineTransform = GXNewTransform();
    GXSkewTransform(italicFace.faceLayer[0].outlineTransform,
italicValue, 0, 0, 0);
    italicFace.faceLayer[0].boldOutset.x = 0;
    italicFace.faceLayer[0].boldOutset.y = 0;

    GXDisposeTransform(italicFace.faceLayer[0].outlineTransform)
}

void DisplayItalicText(void)
{
    gxShape text = GXNewText(6, (unsigned char *)"Italic", nil);

    GXMoveShape(text, 0, ff(20));
    ApplyItalicTextFace(text, -fixed1/6);
    GXDrawShape(text);

    GXMoveShape(text, 0, ff(20));
    ApplyItalicTextFace(text, -fixed1/4);
    GXDrawShape(text);
}
```

Typographic Styles

```

GXMoveShape(text, 0, ff(20));
ApplyItalicTextFace(text, -fixed1/3);
GXDrawShape(text);

GXMoveShape(text, 0, ff(20));
ApplyItalicTextFace(text, -fixed1/2);
GXDrawShape(text);

GXDisposeShape(text);
}

```

Figure 6-10 shows the results of executing the code in Listing 6-2.

Figure 6-10 An italic text face



To create a condensed or extended text face, scale the transform of the text face. The transforms of condensed text faces are scaled to less than 1; those of extended text faces are scaled to more than 1.

```

shrink = ff(60)/100;
face->faceLayer[0].outlineTransform = GXNewTransform();
GXScaleTransform(face->faceLayer[0].outlineTransform, shrink,
                 fixed1, 0, 0);

ScaleMapping(&face->advance mapping, shrink, fix1, 0, 0)

```

The preceding code produces the text face in Figure 6-11.

Figure 6-11 A condensed text face



You can combine these steps to produce a condensed-italic text face. You can also use the transform of a face layer in a text face with more than one layer to produce a drop shadow or other unusual text face.

Listing 6-3 creates a text face with two layers. The first layer is the drop shadow, which is drawn with a closed frame fill; the second layer does not alter the appearance of the text at all. Together, they produce the text face in Figure 6-12.

Listing 6-3 Creating a drop-shadow text face

```
ResetMapping (& layer2map);

layer1map.map[2][0] = fl(0.2);
layer1map.map[2][1] = fl(0.2);

face->faceLayer[0].flags = gxNoAttributes;
face->faceLayer[0].outlineFill = gxClosedFrameFill;
face->faceLayer[0].outlineStyle = nil;
GXSetTransformMapping(face->faceLayer[0].outlineTransform =
    GXNewTransform(), &layer1map);
face->faceLayer[0].boldOutset.x = 0;
face->faceLayer[0].boldOutset.y = 0;

mapping layer1map;

face->faceLayer[1].flags = gxNoAttributes;
face->faceLayer[1].outlineFill = gxWindingFill;
face->faceLayer[1].outlineStyle = nil;
face->faceLayer[1].outlineTransform = nil;
face->faceLayer[1].boldOutset.x = 0;
face->faceLayer[1].boldOutset.y = 0;
```

Figure 6-12 shows the results of executing the code in Listing 6-3.

Figure 6-12 A drop-shadow text face



For more information about the transform object, its properties, and the `GXNewTransform`, `GXSkewTransform`, `GXScaleTransform`, and `GXSetTransformMapping` functions, see *Inside Macintosh: QuickDraw GX Objects*.

The Bold Outset

The `x` and `y` values in the `boldOutset` field of the face layer scale the thicknesses of the glyphs in the displayed shape. The values for bold are treated as though they are for a 1-point font. For example, to get 3 points worth of bold at 48-point text, you set the bold outset to `ff(3)/48`.

You can produce a simple bold text face by setting the `x` value of the `boldOutset` field to any positive value other than 0 and setting the `y` value to 0. In this way, you can create several kinds of boldface text: bold, demibold, black, and so on. See Figure 6-13.

In general, bold fonts tend to get heavy in just the horizontal direction. You can use the `y` bolding as well.

IMPORTANT

You can decrease the amount of bold by using negative values in the bold outset field. ▲

Figure 6-13 Different values of boldface



Setting the Layer Flags

The layer flags primarily allow you to determine the underlining characteristics of the text face. You can also use the layer flags to create a “white layer.”

Listing 6-4 is a sample routine that shows how to create a simple underline text face.

Listing 6-4 Creating a simple underline text face

```
#include "graphics routines.h"
#include "math routines.h"

static ShowSimpleUnderline(void)
{
    gxPoint position = {0, ff(400)};
    gxShape text = GXNewText(9, (unsigned char *)"Underline",
        &position);

    gxTextFace underlineFace = {1, {{{fixed1, 0, 0},
        {0, fixed1, 0}, {0, 0, fract1}}},
        gxUnderlineAdvanceLayer, gxWindingFill, nil, nil,
        {0, 0}};

    GXSetShapeFace(text, &underlineFace);
    GXDrawShape(text);
    GXDisposeShape(text);
}
```

Figure 6-14 shows the results of executing the code in Listing 6-4.

Figure 6-14 A simple underline text face



Listing 6-5 is a sample function that produces a thicker underline text face.

Listing 6-5 Creating a thicker underline

```

void ShowBetterUnderline(void)
{
    gxShape text = GXNewText(9, (unsigned char *)"Underline", nil);
    gxTextFace *underlineFace;
    gxStyle thickPenStyle;
    gxTransform moveDownTransform;

    underlineFace = (gxTextFace *)NewPtr(sizeof(gxTextFace) +
                                          sizeof(gxFaceLayer));

    ResetMapping(&underlineFace->advanceMapping);
    underlineFace->faceLayers = 2;
    underlineFace->faceLayer[0].flags = gxNoAttributes;
    underlineFace->faceLayer[0].outlineFill = gxWindingFill;
    underlineFace->faceLayer[0].outlineStyle = nil;
    underlineFace->faceLayer[0].outlineTransform = nil;
    underlineFace->faceLayer[0].boldOutset.x = 0;
    underlineFace->faceLayer[0].boldOutset.y = 0;

    /* Create a style to thicken the underline. The pen size will
    be scaled by the size of the text drawn, so we make the pen size
    1/12, so that 12 point text gets a 1 pixel underline and larger
    text will get a proportionally thicker underline.
    */
    thickPenStyle = GXNewStyle();
    GXSetStylePen(thickPenStyle, fixed1/12);

    /* Create a transform to position the underline. We want it to
    be 2 pixels below the baseline, so we translate it in the
    positive y direction by 2/12.
    */
    moveDownTransform = GXNewTransform();
    GXMoveTransform(moveDownTransform, 0, fixed1/6);

    underlineFace->faceLayer[1].flags = gxUnderlineAdvanceLayer;
    underlineFace->faceLayer[1].outlineFill = gxNoFill;
    underlineFace->faceLayer[1].outlineStyle = thickPenStyle;
    underlineFace->faceLayer[1].outlineTransform =
        moveDownTransform;
    underlineFace->faceLayer[1].boldOutset.x = 0;
    underlineFace->faceLayer[1].boldOutset.y = 0;

```

Typographic Styles

```

GXSetShapeFace(text, underlineFace);
GXMoveShape(text, 0, ff(400));
GXDrawShape(text);

DisposePtr((void *)underlineFace);
GXDisposeShape(text);
}

```

Figure 6-15 shows the results of executing the code in Listing 6-5.

Figure 6-15 A thicker underline



Setting Text Attributes

By setting the text attributes of a style, you affect how QuickDraw GX treats individual glyphs in a shape.

You should always get the current settings of the text attributes before setting any of them. The `GXSetStyleTextAttributes` function replaces all of the attributes currently associated with the style; if you want any attributes to remain the same, you must include them in the call.

For example, to set the `gxVerticalText` text attribute of a style, you can use the following code:

```

GXSetStyleTextAttributes(myStyle,
    GXGetStyleTextAttributes(myStyle) | gxVerticalText);

```

If you want to clear only that text attribute from a style, you can use the code:

```

GXSetStyleTextAttributes(myStyle,
    GXGetStyleTextAttributes(myStyle) & ~gxVerticalText);

```

Setting the Automatic Text Advance Attribute

By setting the `gxAutoAdvanceText` text attribute you tell QuickDraw GX to take into account changes to the widths of the glyphs in the shape. For example, by applying a bold text face, you increase the widths of the glyphs. If you set the `gxAutoAdvanceText` attribute, QuickDraw GX increases the advance widths of the glyphs by a percentage corresponding to the increase in the filled widths of the bold glyphs.

Typographic Styles

Listing 6-6 is a pair of sample routines that show how the automatic text advance attribute increases the advance width of glyphs in the case of applying a bold text face.

Listing 6-6 Using the automatic text advance attribute

```
void MakeBoldTextFace(gxTextFace* face)
{
    face->faceLayers = 1;
    ResetMapping(&face->advanceMapping);
    face->faceLayer[0].outlineFill = gxWindingFill;
    face->faceLayer[0].flags = 0;
    face->faceLayer[0].outlineStyle = nil;
    face->faceLayer[0].boldOutset.x = fixed1/12;
    face->faceLayer[0].boldOutset.y = fixed1/36;
}

void MakeBoldText(void)
{
    gxShape text;
    gxPoint loc = { ff(50), ff(250);
    gxTextFace myFace;

    text = MakeTextShape("Bolded", "Hoefler Text", ff(200),
                        &loc);
    GXDrawShape(text);

    GXMoveShape(text, 0, ff(200));
    MakeBoldTextFace(&myFace);
    GXSetShapeFace(text, &myFace);
    GXDrawShape(text);

    GXMoveShape(text, 0, ff(200));
    GXSetShapeTextAttributes(text,
                            GXGetShapeStyleAttributes(text) | gxAutoAdvanceText);
    GXDrawShape(text);

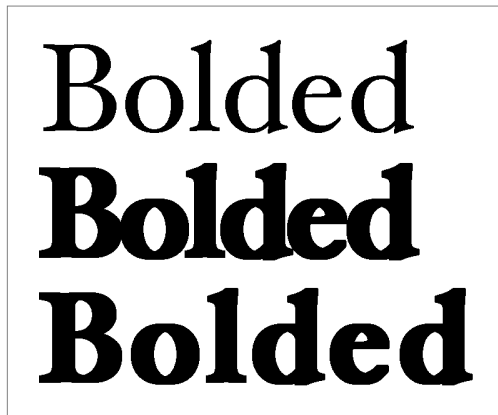
    GXDisposeShape(text);
}
```

Figure 6-16 shows the results of executing the code in Listing 6-6. The same typographic shape is drawn three times. The first time, the shape is drawn with no text face applied. The second time, the shape is drawn with a bold text face, but the `gxAutoAdvanceText`

Typographic Styles

text attribute not set. The third time, the shape is drawn with the text bolded and with the `gxAutoAdvanceText` text attribute set. Note that in the third case the glyph spacing has expanded to account for the bold text face.

Figure 6-16 Drawing text using the `gxAutoAdvanceText` text attribute



Setting the No Contour Grid Attribute

If you set the `gxNoContourGridText` text attribute, QuickDraw GX instructs the scaler *not* to use hints when imaging. If you set this bit, you must also set the `gxNoMetricsGridText` bit.

Listing 6-7 is an example of a routine that shows results of turning off the `gxNoContourGridText` attribute, then turning the shapes into path shapes.

Listing 6-7 Using the no contour grid text attribute

```
void CompareNoContourGrid(void)
{
    gxShape contourGriddedA = GXNewText(1, (unsigned char *)"a",
    nil);
    gxShape noContourGriddedA;

    GXSetShapeTextSize(contourGriddedA, ff(9)); /* A small point
    size increases the effect of gridding */
    GXSetShapePen(contourGriddedA, ff(4));

    /* Make a copy of the original "a" and turn off contour gridding
    */
    noContourGriddedA = GXCopyToShape(nil, contourGriddedA);
    GXSetShapeTextAttributes(noContourGriddedA,
        GXGetShapeTextAttributes(noContourGriddedA) |
        gxNoContourGridText | gxNoMetricsGridText);
}
```

Typographic Styles

```

/* Turn both shapes into paths, and magnify them 75x to show the
differences */
GXSetShapeType(contourGriddedA, gxPathType);
GXSetShapeFill(contourGriddedA, gxClosedFrameFill);
GXSetShapeType(noContourGriddedA, gxPathType);
GXSetShapeFill(noContourGriddedA, gxClosedFrameFill);
GXScaleShape(contourGriddedA, ff(75), ff(75), 0, 0);
GXScaleShape(noContourGriddedA, ff(75), ff(75), 0, 0);
GXMoveShape(contourGriddedA, ff(20), ff(400));
GXMoveShape(noContourGriddedA, ff(400), ff(400));

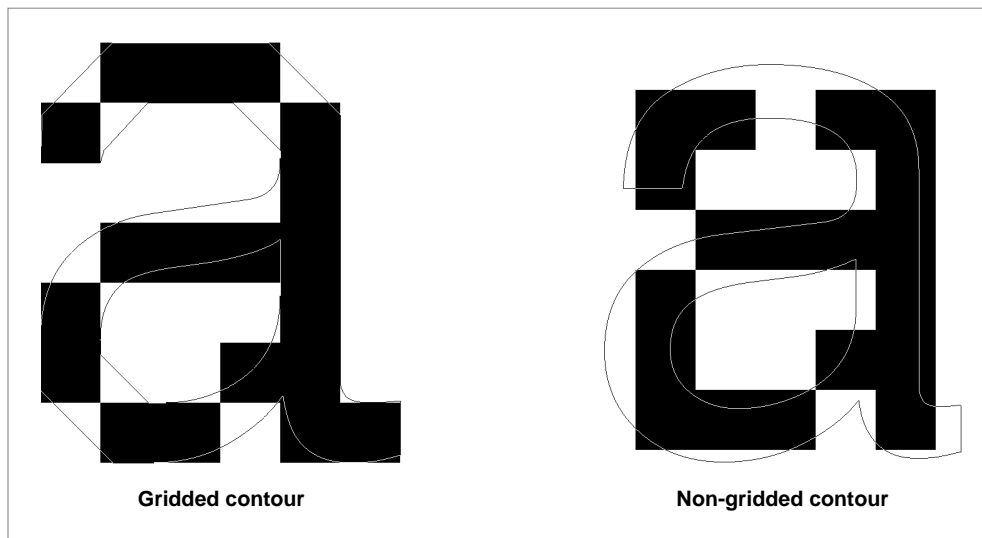
GXDrawShape(contourGriddedA);
GXDrawShape(noContourGriddedA);

GXDisposeShape(contourGriddedA);
GXDisposeShape(noContourGriddedA);
}

```

Figure 6-17 shows the results of executing the code in Listing 6-7. The “a” to the left shows an example with the `gxNoContourGridText` attribute clear, the “a” to right with it set. Note that at small point sizes, more realistic characters can be produced with contour gridding on (with `gxNoContourGridText` clear).

Figure 6-17 Turning the no contour grid attribute off and on



Setting the Vertical Text Attribute

If you set the `gxVerticalText` text attribute in the style, QuickDraw GX returns the font's vertical metrics for the glyphs in the shape when you call a function such as `GXGetGlyphMetrics`. (See the chapter "Typographic Shapes" in this book for a description of this function).

Listing 6-8 is a sample routine that shows how to set the vertical text attribute.

Listing 6-8 Setting the vertical text attribute

```
static void ShowVerticalText(void)
{
    gxShape text = GXNewText(8, (unsigned char *)"Vertical", nil);

    GXMoveShape(text, ff(100), ff(100));
    GXDrawShape(text);

    GXSetShapeTextAttributes(text,
                             GXGetShapeTextAttributes(text) | gxVerticalText);
    GXDrawShape(text);

    GXDisposeShape(text);
}
```

Figure 6-8 on page 6-16 shows the results of the executing code in Listing 6-8.

Depending on the type of shape you're using, setting the `gxVerticalText` text attribute has two very different results. If the shape is a text or glyph shape, QuickDraw GX automatically draws the glyphs that use that style in a vertical line.

Listing 6-9 creates a glyph shape that uses two styles: the first part of the shape uses a Times® Roman style that does not have the vertical text attribute set, and second half of the shape uses a Helvetica style that does.

Listing 6-9 The effects of the vertical text attribute on a glyph shape

```
gxShape      gShape;
gxStyle      helveticaStyle, timesStyle;
gxFont       helveticaFont, timesFont;
static const unsigned char vertText[] = "one way another";
static const short myStyleRuns[] = {8,7};
static gxStyle myStyles[2];
```

Typographic Styles

```

GXFindFonts (gxFullFontName, gxMacintoshPlatform, gxRomanScript,
             gxEnglishLanguage, 9, "Helvetica", 1, 1, &helveticaFont);
helveticaStyle = GXNewStyle ();
GXSetStyleTextSize(helveticaStyle, ff(40));
GXSetStyleFont(helveticaStyle, helveticaFont);
GXSetStyleTextAttributes (helveticaStyle,
                          GXGetStyleTextAttributes (helveticaStyle) | gxVerticalText);

GXFindFonts (gxFullFontName, gxMacintoshPlatform, gxRomanScript,
             gxEnglishLanguage, 9, 11, "Times Roman", 1, 1, &timesFont);
timesStyle = GXNewStyle ();
GXSetStyleTextSize(timesStyle, ff(40));
GXSetStyleFont(timesStyle, timesFont);

GXSetStyleTextAttributes (timesStyle, gxVerticalText);
myStyles[0] = timesStyle;
myStyles[1] = helveticaStyle;
totalLength = sizeof(vertText);

gShape = GXNewGlyphs(totalLength, vertText,
                    nil, nil, nil, myStyleRuns, myStyles);

GXMoveShapeTo(gShape, ff(100), ff(100));

GXDrawShape(gShape);

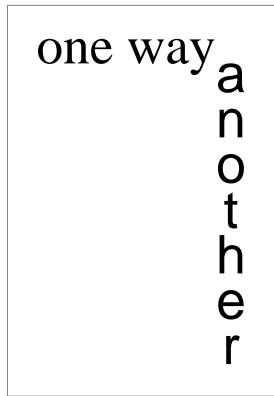
GXDisposeShape(gShape);
GXDisposeStyle(myStyles[0]);

GXDisposeStyle(myStyles[1]);

```

Figure 6-18 shows the results of the code in Listing 6-9. The first half of the shape is drawn in a horizontal line; the second half in a vertical line. Notice that the individual glyphs in the second half of the shape retain their original horizontal orientation but are stacked on top of one another. The glyphs that use the style with the vertical text attribute set are aligned through their centers, and the vertical glyph origin, which is at the top of the glyph “a”) is placed exactly where the advance width of the previous glyph—the space glyph—ends.

Keep in mind, however, that a layout shape describes a single *line* of text. Therefore, when you set the vertical text attribute on a style in a layout shape, QuickDraw GX does not give those glyphs a vertical orientation. It *rotates* them in line and makes them line-up vertically.

Figure 6-18 Using the `gxVerticalText` attribute with a text or glyph shape

For example, you can use the glyph shape in Listing 6-9 to describe the same appearance, using the following code:

```
gxPoint  textPositions = {ff(100), ff(100)};

gShape = GXNewLayout(1, &totalLength, (void *)&vertText,
                    2, myStyleRuns, myStyles,
                    0, nil, nil,
                    nil, &textPositions);
```

Figure 6-19 shows the resulting shape of executing the code. Notice how the glyphs using the Helvetica style, which has the vertical text attribute set, are center aligned just as they are in Figure 6-18, and the vertical glyph origin of the glyph “a” begins where the advance width of the previous space glyph ends.

In the layout shape, however, the entire shape must have one orientation in order to describe a single line. Therefore, in order to orient the second half of the shape correctly, the entire shape needs to be rotated.

Figure 6-19 Using the `gxVerticalText` attribute with a layout shape

See the chapter “Layout Line Control” in this book for more information on creating vertical text in layout shapes.

Applying Patterns and Dashes to Text Faces

You can fill a typographic shape with a pattern, or outline the glyphs in the shape with dashes, just as you can with any other shape. Listing 6-10 is a set of sample routines that creates a repeating sunburst pattern and applies it to a typographic shape through the shape's style object.

Listing 6-10 Filling a typographic shape with a pattern

```
void MakePatternText(void)
{
    gxShape text;
    gxPatternRecord myPat;
    gxPoint loc = { ff(50), ff(200) };

    text = MakeTextShape("Patterned words!", "Hoefler Text",
        ff(144), &loc);
    MakeBurstPattern(&myPat, ff(5), ff(64));
    GXSetShapePattern(text, &myPat);
    GXDisposeShape(myPat.pattern);
    GXDrawShape(text);
    GXDisposeShape(text);
}

/* A utility function that creates a text shape.*/

gxShape MakeTextShape(const char text[], const char font[],
    Fixed textSize, const gxPoint* loc)
{
    gxFont fontID;
    gxShape shape = GXNewText(strlen(text), (unsigned char*)text,
        loc);

    GXFindFonts(nil, gxFullFontName, gxMacintoshPlatform,
        gxRomanScript, gxEnglishLanguage,
        strlen(font), (unsigned char*)font, 1, 1, &fontID);
    GXSetShapeFont(shape, fontID);
    GXSetShapeTextSize(shape, textSize);
    return shape;
}

/* A utility function that creates the sunburst pattern.*/
```

Typographic Styles

```

void MakeBurstPattern(gxPatternRecord* pat,
                    Fixed rotateAmount, Fixed scaleAmount)
{
    gxLine lineData = { { -fixed1, 0 }, { fixed1, 0 } };
    gxShape line = GXNewLine(&lineData);
    gxShape rotated = GXNewShape(gxPolygonType);
    Fixed angle;

    GXSetShapeFill(rotated, gxOpenFrameFill);
    for (angle = 0; angle < ff(180); angle += rotateAmount)
    {
        GXSetShapeParts(rotated, 0, 0, line, gxBreakLeftEdit);
        GXRotateShape(line, rotateAmount, 0, 0);
    }
    GXDisposeShape(line);

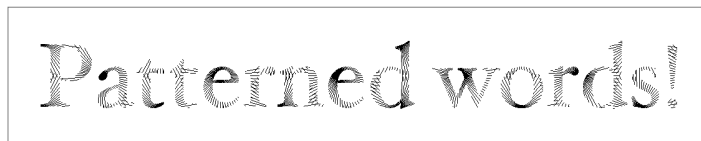
    GXScaleShape(rotated, scaleAmount, scaleAmount, 0, 0);

    pat->attributes = 0;
    pat->pattern = rotated;
    pat->u.x = pat->v.x = MultiplyDivide(scaleAmount, 3, 2);
    pat->u.y = MultiplyDivide(scaleAmount, FixedSquareRoot(ff(3)),
                            ff(2));
    pat->v.y = -pat->u.y;
}

```

Figure 6-20 shows the results of executing the code in Listing 6-10. Note that the sunburst pattern is applied to the shape as a whole, repeating at regular intervals across the text.

Figure 6-20 A typographic shape with a pattern



Creating Unusual Effects With Text Faces

Listing 6-11 is a sample routine that creates a similar sunburst effect, but with a pattern in a text face rather than as a style pattern. This listing uses the same utility functions as does Listing 6-10.

Listing 6-11 Creating an unusual effect

```

void MakePatternTextFace(void)
{
    gxShape text;
    gxPoint loc = { ff(50), ff(200) };
    gxPatternRecord myPat;
    gxTextFace myFace;

    text = MakeTextShape("Patterned words!", "Hoefler Text",
        ff(144), &loc);
    MakeBurstPattern(&myPat, ff(5), fixed1/2);
    MakePatternFace(&myFace, &myPat);
    GXDisposeShape(myPat.pattern);
    GXSetShapeFace(text, &myFace);
    GXDisposeStyle(myFace.faceLayer[0].outlineStyle);
    GXDrawShape(text);
    GXDisposeShape(text);
}

/* A utility function that creates a patterned text shape.*/

void MakePatternFace(gxTextFace* face,
    const gxPatternRecord* pat)
{
    face->faceLayers = 1;
    ResetMapping(&face->advanceMapping);
    face->faceLayer[0].outlineFill = gxWindingFill;
    face->faceLayer[0].flags = 0;
    face->faceLayer[0].outlineStyle = GXNewStyle();
    face->faceLayer[0].outlineTransform = nil;
    face->faceLayer[0].boldOutset.x = 0;
    face->faceLayer[0].boldOutset.y = 0;
    GXSetStyleTextSize(face->faceLayer[0].outlineStyle, fixed1);
    GXSetStylePattern(face->faceLayer[0].outlineStyle, pat);
}

```

Figure 6-21 shows the results of executing the code in Listing 6-11. Note that the sunburst pattern is applied individually to each character; for example, unlike in Figure 6-20, repeated letters are identically patterned wherever they appear.

Figure 6-21 An unusual effect with text faces

Typographic Styles Reference

This section describes the constants and data types, data structures, and functions that are specific to the text shape, glyph shape, and layout shape.

The “Constants and Data Types” section lists the enumerated types that provide information about the shapes and the data structures that contain information about the typographic shapes.

The “Functions” section, beginning on page 6-39, lists the QuickDraw GX functions you use to manipulate the values in the style objects associated with typographic shapes.

The layout shape has its own style and shape routines, in addition to the ones described in this chapter. These routines are described in the chapter “Layout Shapes” in this book.

Constants and Data Types

This section describes the data types that you use to provide information about and retrieve information from the typographic shapes and their associated styles. The data types discussed in this section include

- the `gxTextFace` structure, which you use to define an algorithmically added font style, such as bold or italic, or a more complex font style, such as a special kind of pattern or fill you add to the glyphs of a typographic shape
- the `gxFaceLayer` structure, which you use to describe one of the layers that make up a text face
- the `gxLayerFlags` enumeration, which specifies the characteristics of a face layer in a text face
- the values for alignment, which allow you to pick a predefined alignment setting or create your own
- the `gxTextAttributes` enumeration, which allows you to control how QuickDraw GX alters glyph outlines or sets text to be horizontal or vertical

Text Face

A **text face** is an algorithmically applied typestyle that you define. A text face has both a mapping and face layers. The mapping affects the advance widths and interglyph spacing, but not the shape of the glyphs. The face layers, which are optional, specify the manner in which the glyphs are drawn. All of the face layers are combined to form the visual composite. QuickDraw GX then draws the filled parts of the glyph as it appears with the text face.

The `gxTextFace` structure specifies how to distort the outline, which can be used, among other things, to mimic popular variants of a plain text face.

```
typedef struct {
    long          faceLayers;
    gxMapping      advanceMapping;
    gxFaceLayer    gxFaceLayer[ gxAnyNumber ];
} gxTextFace;
```

Field descriptions

<code>faceLayers</code>	The number of face layers.
<code>advanceMapping</code>	The mapping applied to the advance width for each glyph when the text face is applied. See “Setting the Advance Mapping” on page 6-18.
<code>gxFaceLayer</code>	The face layers attached to this text face. There may be 0 or more face layers, and the first face layer is drawn first. The <code>gxFaceLayer</code> data structure is described in “Face Layers” on page 6-36.

For more information on how to create a text face, see “Creating Text Faces” on page 6-17.

Face Layers

A **face layer** is a description of a part of a text face that you define using the `gxTextFace` structure. The `gxFaceLayer` structure contains the shape fill, the layer flags, a style, a transform, and a degree of boldness that QuickDraw GX should apply. A text face may contain 0 or more face layers.

For more information about how to create a face layer, see “Setting a Face Layer” on page 6-19.

```
typedef struct {
    gxShapeFill    outlineFill;
    gxLayerFlag     flags;
    gxStyle         outlineStyle;
    gxTransform     outlineTransform;
    gxPoint         boldOutset;
} gxFaceLayer;
```

Typographic Styles

Field descriptions

<code>outlineFill</code>	The shape fill for each layer. The possible values for this field are described in <i>Inside Macintosh: QuickDraw GX Graphics</i> . The most useful values with typographic shapes are <code>gxWindingFill</code> and <code>gxClosedFrameFill</code> . The <code>gxEvenOddFill</code> fill may give you unpredictable results.
<code>flags</code>	The flags that describe the drawing and composition of the layers of the text. Possible values for this field are discussed in “Functions” on page 6-39.
<code>outlineStyle</code>	The optional overriding style for the face. The style’s pen size is scaled by a factor corresponding to the point size of the text. This style cannot have its own text face; if so QuickDraw GX posts an error.
<code>outlineTransform</code>	The matrix and clip used to distort the glyph. This field governs the algorithmic application of the italic, condensed, and extended typesstyles as well as any designs and patterns in the text face. All distortions are scaled factors corresponding to text point size and by any mappings affecting the text. This clip has a more global effect than the clip specified in the layer flags.
<code>boldOutset</code>	The degree of boldness QuickDraw GX should apply in the x and y directions for 1-point text.

Layer Flags

The layer flags describe the characteristics of one layer of a text face. For more information about the meanings of the flags, see “Setting the Layer Flags” on page 6-23.

```
enum gxLayerFlags{
    gxUnderlineAdvanceLayer    = 0x0001,
    gxSkipWhiteSpaceLayer     = 0x0002,
    gxUnderlineIntervalLayer   = 0x0004,
    gxUnderlineContinuationLayer= 0x0008,
    gxWhiteLayer               = 0x0010,
    gxClipLayer                = 0x0020,
    gxStringLayer              = 0x0040
};

typedef long gxLayerFlag;
```

Constant descriptions

<code>gxUnderlineAdvanceLayer</code>	Draws an underline from the beginning of the text that shares one text face to the end, including white spaces. This bit must be set if the <code>gxSkipWhiteSpaceLayer</code> , <code>gxUnderlineIntervalLayer</code> , or <code>gxUnderlineContinuationLayer</code> bit is set.
--------------------------------------	---

Typographic Styles

<code>gxSkipWhiteSpaceLayer</code>	Does not draw an underline with glyphs that have no contours (such as the space character) if the <code>gxUnderlineAdvanceLayer</code> bit is also set.
<code>gxUnderlineIntervalLayer</code>	Draws an underline through the gaps between text of different text faces. If you set this bit, you must also set the <code>gxStringLayer</code> bit.
<code>gxUnderlineContinuationLayer</code>	Draws an underline across text of different style runs. If you set this bit, you must also set the <code>gxStringLayer</code> bit. Also, the previous style in the shape must have <code>gxUnderlineAdvanceLayer</code> set; if it doesn't, you will get an error.
<code>gxWhiteLayer</code>	Erases portions of previously drawn layers. You can only set this bit for the second or greater layer.
<code>gxClipLayer</code>	Clips the layer to the original outline of the glyph. You should set this bit if you want to clip a pattern or fill to the text.
<code>gxStringLayer</code>	Connects the text of different text faces and style runs.

Alignment Values

Several values are predefined for various types of alignment.

```
#define gxLeftJustify    0
#define gxCenterJustify (fract1/2)
#define gxRightJustify  fract1
#define gxFillJustify   -1
```

Constant descriptions

`gxLeftJustify` Draws left-aligned text.
`gxCenterJustify` Draws centered text.
`gxRightJustify` Draws right-aligned text.
`gxFillJustify` Draws fully justified text.

For more information see “Alignment” on page 6-11.

Text Attributes

Each style object has a set of **text attributes**, which consist of a group of flags that modify the behavior of the style object associated with typographic shapes. These flags allow you to specify how QuickDraw GX alters glyph outlines or chooses the proper types of metrics for horizontal or vertical text. These flags are defined in the `gxTextAttributes` enumeration.

```
enum gxTextAttributes{
    gxAutoAdvanceText      = 0x0001,
    gxNoContourGridText    = 0x0002,
    gxNoMetricsGridText    = 0x0004,
```

Typographic Styles

```

    gxAnchorPointsText    = 0x0008,
    gxVerticalText        = 0x0010,
    gxNoOpticalScaleText  = 0x0020
};

```

Constant descriptions**gxAutoAdvanceText**

Tells QuickDraw GX to take into account changes to the widths of glyphs in the shape.

gxNoContourGridText

Prevents QuickDraw GX from using hinted outlines before displaying the text.

gxNoMetricsGridText

Tells QuickDraw GX to use the ideal metrics to space the glyphs in a typographic shape.

gxAnchorPointsText

Tells QuickDraw GX to include data for all the outline's points. QuickDraw GX's default action is to return a basic outline, removing points that do not affect the shape, such as single-point contours.

gxVerticalText

Tells QuickDraw GX to return the vertical advance and side bearing metrics. The default values are the horizontal advance and side bearing metrics.

gxNoOpticalScaleText

Disables QuickDraw GX's attempt to automatically set the optical scale variation value.

Text attributes are described in "Text Attributes" on page 6-14.

Functions

This section describes the routines that access, retrieve, change, or delete information about the typographic shapes.

You can retrieve and set basic shape and style information, such as the text attributes, the text face, the font, the text size, the justification amount, the platform, or the font variation descriptions.

Getting and Setting the Font of a Style Object

The font property of style objects specifies the font (or font family) of a style object. You use the `gxFont` data structure, which is described in the chapter "Fonts Objects" of this book, when retrieving or setting the font used by a style object.

You can use the `GXGetStyleFont` function to retrieve the font information from a style object and the `GXSetStyleFont` function to specify the font information for a style object.

The `GXGetShapeFont` and `GXSetShapeFont` functions provide a way to retrieve and specify the font information for the style object associated with a particular shape.

GXGetStyleFont

You can use the `GXGetStyleFont` function to determine the font currently set by a style object.

```
gxFont GXGetStyleFont(gxStyle source);
```

`source` A reference to the style object whose font you want to determine

function result The font associated with the style object.

DESCRIPTION

The `GXGetStyleFont` function returns the font associated with the style object. To get the name of the font, you can use the `GXFindFontName` function.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

SEE ALSO

The `GXGetStyleFace` function (page 6-43) determines the text face currently set by a style object.

Fonts are discussed in the chapter “Font Objects” in this book.

GXSetStyleFont

You can use the `GXSetStyleFont` function to set or change the font used by a style object.

```
void GXSetStyleFont(gxStyle target, gxFont aFont);
```

`target` A reference to the style object whose font you want to set or change.
`aFont` The new font.

DESCRIPTION

The `GXSetStyleFont` function sets the font used by the style object specified by the `target` parameter. If you want the default font, pass `nil` in the `aFont` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
 style_is_nil
 illegal_font_parameter

Notices (debugging version)

font_already_set

SEE ALSO

Fonts are discussed in the chapter “Font Objects” in this book.

GXGetShapeFont

You can use the `GXGetShapeFont` function to determine the font set for the style object of a particular QuickDraw GX typographic shape.

```
gxFont GXGetShapeFont(gxShape source);
```

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

function result The font of the style object of a shape.

DESCRIPTION

The `GXGetShapeFont` function returns as the function result the font of the style object of a shape.

The function returns information about the font setting of the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleFont` (page 6-40) and `GXSetStyleFont` (page 6-40) functions to determine the fonts of the styles stored in the shape’s geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
 shape_is_nil

SEE ALSO

Fonts are discussed in the chapter “Font Objects” in this book.

GXSetShapeFont

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

```
void GXSetShapeFont(gxShape target, gxFont aFont);
```

`target` A reference to the shape whose font you want to alter.

`aFont` The new font.

DESCRIPTION

The `GXSetShapeFont` function sets the font of the style object associated with the shape specified by the `target` parameter. If you want the default font, pass `nil` in the `aFont` parameter. If a style object is shared among shapes, `GXSetShapeFont` copies the style object so that only the shape in the `target` parameter is affected by the changes to the font.

This function provides a convenient way to change the font of a shape without having to call the `GXGetShapeStyle` function to obtain a reference to the shape's style object.

The function specifies the font for the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleFont` (page 6-40) and `GXSetStyleFont` (page 6-40) functions to get and set the fonts of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`shape_is_nil`

`illegal_font_parameter`

Notices (debugging version)

`font_already_set`

SEE ALSO

Fonts are discussed in the chapter "Font Objects" in this book.

Getting and Setting the Text Face

The text face property of a style object specifies an algorithmic typestyle that you want to apply to text. You use the `gxTextFace` data structure, which is described on page 6-36, when retrieving or setting text face information.

You can use the `GXGetStyleFace` function to retrieve the text face information from a style object and the `GXSetStyleFace` function to specify the text face information for a style object.

Typographic Styles

The `GXGetShapeFace` and `GXSetShapeFace` functions provide a way to retrieve and specify the text face information for the style object associated with a particular shape.

GXGetStyleFace

You can use the `GXGetStyleFace` function to get the text face currently set in a style object.

```
long GXGetStyleFace(gxStyle source, gxTextFace *face);
```

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

face The text face set in the style object, returned by the function. This can be `nil`, which just returns the layer count.

function result The number of layers in the text face. If there is no text face, the function returns `-1`.

DESCRIPTION

The `GXGetStyleFace` function returns the number of layers in the text face. The function also returns, in the `face` parameter, the text face used by the style object.

Note

Your application must allocate enough memory to store the text face and all of the face layers of that text face. See “Creating Text Faces” on page 6-17. ♦

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

GXSetStyleFace

You can use the `GXSetStyleFace` function to set or change the text face of a style object.

```
void GXSetStyleFace(gxStyle target, const gxTextFace *face);
```

target A reference to the style object whose text face you want to change.

face The new text face for the style object. If `nil`, it will remove an existing face.

Typographic Styles

DESCRIPTION

The `GXSetStyleFace` function sets the text face of the style object specified by `target` to the text face specified in the `face` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

<code>out_of_memory</code>	
<code>style_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>inverseFill_must_set_cliplayer_flag</code>	(debugging version)
<code>style_wrong_type</code>	(debugging version)
<code>layer_style_cannot_contain_face</code>	(debugging version)
<code>transform_wrong_type</code>	(debugging version)

Notices (debugging version)

`face_already_set`

GXGetShapeFace

You can use the `GXGetShapeFace` function to determine the text face set for the style object of a particular QuickDraw GX typographic shape.

```
long GXGetShapeFace(gxShape source, gxTextFace *face);
```

`source` A reference to the shape whose text face you want to determine.

`face` The text face of the shape, returned by the function.

function result The number of layers in the text face. If there is no text face, the function returns -1.

DESCRIPTION

The `GXGetShapeFace` function returns the number of layers in the text face named by the `face` parameter. The function also returns the text face itself.

The function returns information about the text face setting of the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleFace` function, described on page 6-43 and the `GXSetStyleFace` function, described on page 6-43, to determine the text faces for the styles stored in the shape's geometry.

Note

Your application must allocate enough memory to store the text face and all of the face layers of that text face. See "Creating Text Faces" on page 6-17. ♦

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory
shape_is_nil

GXSetShapeFace

You can use the `GXSetShapeFace` function to alter the text face of the style object associated with a particular shape.

```
void GXSetShapeFace(gxShape target, const gxTextFace *face);
```

`target` A reference to the shape whose text face you want to alter.
`face` The new text face.

DESCRIPTION

The `GXSetShapeFace` function sets the text face of the style object associated with the shape specified by the `target` parameter. If a style object is shared among shapes, `GXSetShapeFace` copies the style object so that only the shape in the parameter is affected by the changes to the text face.

This function provides a convenient way to change the text face of a shape without calling the `GXGetShapeStyle` function to obtain a reference to the shape's style object.

You can use this function in combination with the `GXGetShapeFace` function, described on page 6-44, to set or clear single style attributes.

ERRORS, WARNINGS, AND NOTICES

Errors

out_of_memory	
style_is_nil	
parameter_out_of_range	(debugging version)
inverseFill_must_set_cliplayer_flag	(debugging version)
style_wrong_type	(debugging version)
layer_style_cannot_contain_face	(debugging version)
transform_wrong_type	(debugging version)

Notices (debugging version)

face_already_set

Getting and Setting the Text Size of a Style Object

The text size property of style objects specifies the size, in typographic points, of the text of a style object.

You can use the `GXGetStyleTextSize` function to retrieve the text size from a style object and the `GXSetStyleTextSize` function to specify the text size of a style object.

The `GXGetShapeTextSize` and `GXSetShapeTextSize` functions provide a way to retrieve and specify the text size for the style object associated with a particular shape.

GXGetStyleTextSize

You can use the `GXGetStyleTextSize` function to determine the text size of a style object.

```
Fixed GXGetStyleTextSize(gxStyle source);
```

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

function result The text size, in typographic points, of the style object named by the `source` parameter.

DESCRIPTION

The `GXGetStyleTextSize` function returns the text size, in typographic points, of the style object named by the `source` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

GXSetStyleTextSize

You can use the `GXSetStyleTextSize` function to set or change the text size used by a style object.

```
void GXSetStyleTextSize(gxStyle target, Fixed size);
```

target A reference to the style object whose text size you want to set.

size The new text size, in points. This parameter can be any positive value, including fractional sizes. A value of 0 resets the text size to the point size natural for the current script. For Roman scripts, this is 12 points.

DESCRIPTION

The `GXSetStyleTextSize` function sets the text size, in typographic points, of the style object.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`style_is_nil`
`parameter_out_of_range` (debugging version)

Notices (debugging version)

`text_size_already_set`

GXGetShapeTextSize

You can use the `GXGetShapeTextSize` function to determine the text size, in typographic points, set for the style object of a particular QuickDraw GX typographic shape.

```
FixedGXGetShapeTextSize(gxShape source);
```

`source` A reference to the shape whose text size you want to determine.

function result The text size, in typographic points, of the style object of the shape named by the `source` parameter.

DESCRIPTION

The `GXGetShapeTextSize` function returns the text size, in typographic points, of the style object of the shape named by the `source` parameter.

The function returns information about the text size setting of the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleTextSize` (page 6-46) and `GXGetStyleTextSize` (page 6-46) functions to set the sizes of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`style_is_nil`

GXSetShapeTextSize

You can use the `GXSetShapeTextSize` function to alter the text size of the style object associated with a particular shape.

```
void GXSetShapeTextSize(gxShape target, Fixed size);
```

<code>target</code>	A reference to the shape whose text size you want to change.
<code>size</code>	The new text size, in points. This parameter can be any positive value, including fractional sizes. A value of 0 resets the text size to the point size natural for the current script. For Roman scripts, this is 12 points.

DESCRIPTION

The `GXSetShapeTextSize` function sets the text size, in typographic points, of the style object associated with the shape specified by the `target` parameter. If a style object is shared among shapes, `GXSetShapeTextSize` copies the style object so that only the shape in the `target` parameter is affected by the changes to the text size.

This function provides a convenient way to change the text size of a shape without calling the `GXGetShapeStyle` function to obtain a reference to the shape's style object.

ERRORS, WARNINGS, AND NOTICES

Errors

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

Notices (debugging version)

<code>text_size_already_set</code>

Getting and Setting the Alignment of a Style Object

The alignment property of style objects specifies the type of alignment used in a style object. For more information, see “Alignment” on page 6-11. Possible values for alignment are described in “Alignment Values” on page 6-38.

You can use the `GXGetStyleJustification` function to retrieve the alignment amount from a style object and the `GXSetStyleJustification` function to specify the alignment for a style object.

The `GXGetShapeJustification` and `GXSetShapeJustification` functions provide a way to retrieve and specify the alignment of the style object associated with a text or glyph shape.

The justification value accounts for the difference between ideal and device metrics when the text is measured or drawn. For more control over how justification is used in your text, see the chapter “Layout Line Control” in this book.

GXGetStyleJustification

You can use the `GXGetStyleJustification` function to determine the alignment set by a style object.

```
fract GXGetStyleJustification(gxStyle source);
```

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

function result The alignment set by a style object

DESCRIPTION

The `GXGetStyleJustification` function returns the alignment set by a style object. Possible values for the style's alignment are discussed in "Alignment Values" on page 6-38.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`style_is_nil`

GXSetStyleJustification

You can use the `GXSetStyleJustification` function to set or change the alignment used by a style object.

```
void GXSetStyleJustification(gxStyle target, fract justify);
```

target A reference to the style object whose alignment you want to change.

justify The alignment you want to set in the style object. Possible values are discussed in "Alignment Values" on page 6-38.

DESCRIPTION

The `GXSetStyleJustification` function sets the alignment used by the style object named by the `target` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`style_is_nil`

`parameter_out_of_range`

(debugging version)

Notices (debugging version)

`justification_already_set`

GXGetShapeJustification

You can use the `GXGetShapeJustification` function to determine the alignment set for the style object of a particular QuickDraw GX typographic shape.

```
fractGXGetShapeJustification(gxShape source);
```

`source` A reference to the shape whose set alignment you want to determine.

function result The alignment set in the style object used by the specified shape.

DESCRIPTION

The `GXGetShapeJustification` function returns the alignment set in the style object associated with the shape specified by `source`.

The function returns information about the alignment setting of the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleJustification` (page 6-49) and `GXGetStyleJustification` (page 6-49) functions to determine the alignments of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

GXSetShapeJustification

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

```
void GXSetShapeJustification(gxShape target, fract justify);
```

`target` A reference to the shape whose alignment you want to alter.

`justify` The alignment you want to set in this shape. Possible values for this parameter are described in "Alignment Values" on page 6-38.

DESCRIPTION

The `GXSetShapeJustification` function sets the alignment of the style object associated with the shape specified by the `target` parameter. If a style object is shared among shapes, `GXSetShapeJustification` copies the style object so that only the shape in the `target` parameter is affected by the changes to the alignment value.

This function provides a convenient way to change the alignment of a shape without calling the `GXGetShapeStyle` function to obtain a reference to the shape's style object.

Typographic Styles

The function returns information about the alignment setting of the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleJustification` (page 6-49) and `GXSetStyleJustification` (page 6-49) functions to set the alignment of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`
`parameter_out_of_range`

Getting and Setting the Font Variations of a Style Object

The font variations property of style objects specifies a particular variation coordinate to be used with the style's font. You use the `gxFontVariation` data type, which is described in the chapter "Fonts Objects" of this book, when retrieving or setting font variations for the user.

You can use the `GXGetStyleFontVariations` function to retrieve the font variations from a style object and the `GXSetStyleFontVariations` function to specify the font variations for a style object.

The `GXGetShapeFontVariations` and `GXSetShapeFontVariations` functions provide a way to retrieve and specify the font variations for the style object associated with a particular shape.

You should use the font variation functions described in the chapter "Fonts Objects" to retrieve, add, or change font variation data in the font.

GXGetStyleFontVariations

You can use the `GXGetStyleFontVariations` function to determine the font variations set by a style object.

```
long GXGetStyleFontVariations(gxStyle source,
                              gxFontVariation variations[]);
```

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

variations The font variations set by the style object, returned by the function, allocated by the application.

function result The number of font variations in the style object. The function returns 0 if there are no font variations.

The `GXGetStyleFontVariations` function returns the number of font variations associated with the style object. It returns the font variations in the `variations` parameter, if it is not set to `nil`.

Errors

```
out_of_memory
style_is_nil
```

Font variations are described in the chapter “Fonts Objects” in this book.

You can use the `GXSetFontVariations` function to set or change the font variation used by a style object.

target	A reference to the style object whose font variation you want to change.
count	The number of font variations you are specifying.
variations	The new font variation for the style object.

The `GXSetStyleFontVariations` function sets the font variation of a style object.

```

Errors
out_of_memory
style_is_nil
parameter_out_of_range                                     (debugging version)

Notices (debugging version)
font_variations_already_set

```

Font variations are described in the chapter “Fonts Objects” in this book.

GXGetShapeFontVariations

You can use the `GXGetShapeFontVariations` function to determine the font variations set for the style object of a particular QuickDraw GX typographic shape.

```
long GXGetShapeFontVariations(gxShape source,
                              gxFontVariation variations[]);
```

`source` A reference to the shape whose font variations you want to determine.

`variations` The font variations of the shape, returned by the function if it is not nil.

function result The number of font variations in the style object associated with the shape object. The function returns 0 if there are no font variations.

DESCRIPTION

The `GXGetShapeFontVariations` function returns the number of font variations associated with the style object of the shape object. The function also returns the font variations.

The function returns information about the font variation setting of the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleFontVariations` (page 6-51) and `GXSetStyleFontVariations` (page 6-52) functions to get the font variation of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

SEE ALSO

Font variations are described in the chapter “Fonts Objects” in this book.

GXSetShapeFontVariations

You can use the `GXSetShapeFontVariations` function to alter the font variations associated with a particular shape.

```
void GXSetShapeFontVariations(gxShape target, long count,
                             const gxFontVariation variations[]);
```

`target` A reference to the shape whose font variations you want to alter.

`count` The number of font variations you are specifying.

`variations` The new font variations.

DESCRIPTION

The `GXSetShapeFontVariations` function sets the font variations of the style object associated with the shape specified by the `target` parameter. If a style object is shared among shapes, `GXSetShapeFontVariations` copies the style object so that only the shape in the `target` parameter is affected by the changes to the font variations.

This function provides a convenient way to change the font variations of a shape without calling the `GXGetShapeStyle` function to obtain a reference to the shape's style object.

The function specifies the font variation setting of the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleFontVariations` (page 6-51) and `GXSetStyleFontVariations` (page 6-52) functions to set the font variations of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`

`style_is_nil`

`parameter_out_of_range`

(debugging version)

Notices (debugging version)

`font_variations_already_set`

SEE ALSO

Font variations are described in the chapter “Fonts Objects” in this book.

Retrieving the Elements in a Font Variation Suite

QuickDraw GX allows you to retrieve all the elements in a font variation suite. The font variation suite contains complete information on the font variations specified for a shape or style object. You can

- retrieve the number of elements in a font variation suite for a style object with the `GXGetStyleFontVariationSuite` function
- retrieve the font variation suite for the style object associated with a specified shape with the `GXGetShapeFontVariationSuite` function

The font variation suite is described in “Font Variations” on page 6-13. Font variations are described in the chapter “Font Objects” in this book.

GXGetStyleFontVariationSuite

You can use the `GXGetStyleFontVariationSuite` function to retrieve the font variation suite for a style object.

```
long GXGetStyleFontVariationSuite(gxStyle source,
                                   gxFontVariation variations[]);
```

source A reference to the style object whose font variation suite you want to retrieve.

variations An array of font variation structures. On return this array contains the font variations for the style.

function result The number of elements in the font variation suite, which is equal to the number of variation axes in a font.

DESCRIPTION

The `GXGetStyleFontVariationSuite` function returns the number of elements in the font variation suite for the font specified by the style associated with the source object. It returns the variations themselves in the `variations` parameter. If you pass `nil` for the `variations` parameter, this function returns a valid result but returns no array. You typically call this function twice, first with a `nil` value for `variations` in order to get the right size of array to allocate and the second time to retrieve the array itself.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

SEE ALSO

To get the font variation suite for a shape object, use the `GXGetShapeFontVariationSuite` function, described in the next section.

Font Variations, including `GXCountFontVariations`, are described in the chapter “Font Objects” in this book.

GXGetShapeFontVariationSuite

You can use the `GXGetShapeFontVariationSuite` function to retrieve the font variation suite for the style object associated with a specified shape.

```
long GXGetShapeFontVariationSuite(gxShape source,
                                   gxFontVariation variations[]);
```

source A reference to the shape object whose style object’s font variation suite you want to retrieve.

variations An array of font variation structures. On return this array contains the font variations for the style.

function result The number of elements in the font variation suite. The function returns 0 if there are no font variations.

DESCRIPTION

The `GXGetShapeFontVariationSuite` function returns the number of elements in the font variation suite for the font specified by the style object associated with the source shape. It returns the variations themselves in the `variations` parameter. If you pass `nil` for the `variations` parameter, this function returns a valid result but returns no array. You typically call this function twice, first with a `nil` value for `variations` in order to get the right size of array to allocate and the second time to retrieve the array itself.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`shape_is_nil`

SEE ALSO

To get the font variation suite from a shape object, use the `GXGetStyleFontVariationSuite` function, described on page 6-55.

Font Variations are described in the chapter “Font Objects” in this book.

Retrieving Font Metrics

QuickDraw GX allows you to retrieve font metrics for an object in several ways. You can

- retrieve the font metrics for a style object, including line spacing and caret angle, with the `GXGetStyleFontMetrics` function
- retrieve the font metrics for a style object associated with a shape with the `GXGetShapeFontMetrics` function
- retrieve the font metrics for a style object associated with a shape, taking into account the shape's transform, with the `GXGetShapeLocalFontMetrics` function
- retrieve the font metrics for a style object associated with a shape, taking into account the mappings on a specified view port and view device, with the `GXGetShapeDeviceFontMetrics` function

Font metrics are described in “Font Metrics” on page 6-14.

GXGetStyleFontMetrics

You can use the `GXGetStyleFontMetrics` function to retrieve the font metrics for a style object, including line spacing and caret angle.

```
void GXGetStyleFontMetrics(gxStyle sourceStyle, gxPoint* before,
                           gxPoint* after, gxPoint* caretAngle,
                           gxPoint* caretOffset);
```

`sourceStyle`

A reference to the style object whose font metrics you need.

`before`

A pointer to a `gxPoint` structure. On return, the point specifies the distance and direction to the previous line of text. For horizontal text, this corresponds to the font's ascent.

`after`

A pointer to a `gxPoint` structure. On return, the point specifies the distance and direction to the next line of text. For horizontal text, this corresponds to the font's descent.

`caretAngle`

A pointer to a `gxPoint` structure. On return, the point specifies the direction for the text selection caret.

`caretOffset`

A pointer to a `gxPoint` structure. On return, the point specifies the direction and distance relative to the origin, where the text selection caret should intersect the baseline of the text.

DESCRIPTION

The `GXGetStyleFontMetrics` function returns the font metrics of a style object, including its line spacing and its caret angle. These values are returned as points representing the vectors. The values of the vectors differentiate between horizontal and vertical text. This function takes into account the point size, variations, and the settings of the `gxNoMetricsGridText` and `gxVerticalText` text attributes.

ERRORS, WARNINGS, AND NOTICES

Errors

`illegal_type_for_shape` (if not typographic) (debugging version)
`shape_is_nil`

SEE ALSO

The `GXGetShapeFontMetrics` function is described in the next section. The `GXGetShapeLocalFontMetrics` function is described on page 6-59.

The `GXGetShapeDeviceFontMetrics` function is described on page 6-60.

GXGetShapeFontMetrics

You can use the `GXGetShapeFontMetrics` function to retrieve the font metrics for the style object associated with a shape.

```
void GXGetShapeFontMetrics(gxShape source, gxPoint* before,
                           gxPoint* after, gxPoint* caretAngle,
                           gxPoint* caretOffset);
```

<code>source</code>	A reference to the shape object whose font metrics you need.
<code>before</code>	A pointer to a <code>gxPoint</code> structure. On return, the point specifies the distance and direction to the previous line of text. For horizontal text, this corresponds to the font's ascent.
<code>after</code>	A pointer to a <code>gxPoint</code> structure. On return, the point specifies the distance and direction to the next line of text. For horizontal text, this corresponds to the font's descent.
<code>caretAngle</code>	A pointer to a <code>gxPoint</code> structure. On return, the point specifies the direction for the text selection caret.
<code>caretOffset</code>	A pointer to a <code>gxPoint</code> structure. On return, the point specifies the direction and distance relative to the shape's origin, where the text-selection caret should intersect the baseline of the text.

DESCRIPTION

The `GXGetShapeFontMetrics` function returns the font metrics of the `style` object associated with the specified shape, including its line spacing and its caret angle. These values are returned as points representing vectors. The values of the vectors differentiate between horizontal and vertical text. This function is equivalent to calling the `GXGetStyleFontMetrics` function with the result of `GXGetShapeStyle`.

ERRORS, WARNINGS, AND NOTICES

Errors

`illegal_type_for_shape` (if not typographic) (debugging version)
`shape_is_nil`

SEE ALSO

The `GXGetStyleFontMetrics` function is described on page 6-57. The `GXGetShapeLocalFontMetrics` function is described in the next section. The `GXGetShapeDeviceFontMetrics` function is described on page 6-60.

GXGetShapeLocalFontMetrics

You can use the `GXGetShapeLocalFontMetrics` function to retrieve the font metrics for the style object associated with a shape, taking into account the shape's transform. The results are expressed in local coordinates.

```
void GXGetShapeLocalFontMetrics(gxShape sourceShape,
                                gxPoint* before,
                                gxPoint* after,
                                gxPoint* caretAngle,
                                gxPoint* caretOffset);
```

`sourceShape`

A reference to the shape object whose font metrics you need.

`before`

A pointer to a `gxPoint` structure. On return, the point specifies the distance and direction to the previous line of text. For horizontal text, this corresponds to the font's ascent. If the parameter is `nil`, it is ignored.

`after`

A pointer to a `gxPoint` structure. On return, the point specifies the distance and direction to the next line of text. For horizontal text, this corresponds to the font's descent. If the parameter is `nil`, it is ignored.

`caretAngle`

A pointer to a `gxPoint` structure. On return, the point specifies the direction for the text-selection caret. If the parameter is `nil`, it is ignored.

`caretOffset`

A pointer to a `gxPoint` structure. On return, the point specifies the direction and distance relative to the shape's origin, where the text selection caret should intersect the baseline of the text. If the parameter is `nil`, it is ignored.

Typographic Styles

DESCRIPTION

The `GXGetShapeLocalFontMetrics` function returns the font metrics for the style object associated with the source shape, in local space. These values are returned as points that represent vectors. The values of the vectors differentiate between horizontal and vertical text and account for any mapping that may be applied to the shape through its transform object.

ERRORS, WARNINGS, AND NOTICES

Errors

`illegal_type_for_shape` (if not typographic) (debugging version)
`shape_is_nil`

SEE ALSO

The `GXGetStyleFontMetrics` function is described on page 6-57. The `GXGetShapeFontMetrics` function is described on page 6-58. The `GXGetShapeDeviceFontMetrics` function is described in the next section.

GXGetShapeDeviceFontMetrics

You can use the `GXGetShapeDeviceFontMetrics` function to retrieve the font metrics for the style object associated with a shape, taking into account the mappings on the specified view port and view device. The result is expressed in device coordinates.

```
void GXGetShapeDeviceFontMetrics(gxShape sourceShape,
                                gxViewPort port,
                                gxViewDevice device,
                                gxPoint* before,
                                gxPoint* after,
                                gxPoint* caretAngle,
                                gxPoint* caretOffset);
```

`sourceShape`

A reference to the shape object whose font metrics you need.

`port`

The view port whose mappings you need to take into account.

`device`

The view device whose mappings you need to take into account.

`before`

A pointer to a `gxPoint` structure. On return, the point specifies the distance and direction to the previous line of text. For horizontal text, this corresponds to the font's ascent.

`after`

A pointer to a `gxPoint` structure. On return, the point specifies the distance and direction to the next line of text. For horizontal text, this corresponds to the font's descent.

Typographic Styles

caretAngle A pointer to a `gxPoint` structure. On return, the point specifies the direction for the text-selection caret.

caretOffset

A pointer to a `gxPoint` structure. On return, the point specifies the direction and distance relative to the shape's origin, where the text-selection caret should intersect the baseline of the text.

DESCRIPTION

The `GXGetShapeDeviceFontMetrics` function returns the font metrics for the style object associated with the source shape in device space. These values are returned as points that represent vectors. The values of the vectors differentiate between horizontal and vertical text and account for any mapping that may be on the shape's transform as well as the specified view port and view device.

ERRORS, WARNINGS, AND NOTICES

Errors

`illegal_type_for_shape` (if not typographic) (debugging version)
`shape_is_nil`

SEE ALSO

The `GXGetStyleFontMetrics` function is described on page 6-57. The `GXGetShapeFontMetrics` function is described on page 6-58. The `GXGetShapeLocalFontMetrics` function is described on page 6-59.

Getting and Setting the Encoding of a Style Object

The encoding in a style is the combination of the platform, script, and language. Platforms, scripts, and languages are described in the chapter “Fonts Objects” in this book.

You can use the `GXGetStyleEncoding` function to retrieve the platform, script, and language information from a style object and the `GXSetStyleEncoding` function to specify the platform, script, and language of a style object.

The `GXGetShapeEncoding` and `GXSetShapeEncoding` functions provide a way to retrieve and specify the platform, script, and language for the style object associated with a particular shape.

Text shapes contain only one encoding value. Glyphs shapes and layout shapes may contain multiple encoding values, because these shapes may contain several different styles in the styles arrays in the shapes' geometries. Each style can contain a separate encoding value.

GXGetStyleEncoding

You can use the `GXGetStyleEncoding` function to determine the platform, script, and language of a style object.

```
gxFontPlatform GXGetStyleEncoding(gxStyle source,
                                   gxFontScript *script,
                                   gxFontLanguage *language);
```

<code>source</code>	A reference to the style object whose platform, script, or language you want to determine.
<code>script</code>	The style object's script, returned by the function.
<code>language</code>	The style object's language, returned by the function.

function result The platform of the style object.

DESCRIPTION

The `GXGetStyleEncoding` function returns the style's platform, script, and language.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

SEE ALSO

The `gxFontPlatform`, `gxFontScript`, and `gxFontLanguage` data types are discussed in the chapter "Fonts Objects" of this book.

GXSetStyleEncoding

You can use the `GXSetStyleEncoding` function to set or change the platform, script, and language of a style object.

```
void GXSetStyleEncoding(gxStyle target, gxFontPlatform platform,
                        gxFontScript script,
                        gxFontLanguage language);
```

<code>target</code>	A reference to the style object whose platform, script, or language you want to change.
<code>platform</code>	The new platform value.

Typographic Styles

`script` The new script value.
`language` The new language value.

DESCRIPTION

The `GXSetStyleEncoding` function sets the platform, script, and language of the style object. Default values for the platform, script, and language are `gxMacintoshPlatform`, `gxRomanScript`, and `gxNoLanguage`, respectively.

The `GXSetStyleEncoding` function does not change or translate the character codes in the shape. You cannot use `GXSetStyleEncoding` to convert a particular character from one encoding to another, because a character found on one platform may not be represented on another platform.

If you set the platform to the value `gxGlyphPlatform`, you should set the values of the script and language parameters to `gxNoScript` and `gxNoLanguage`.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`
`inconsistent_parameters` (debugging version)

Notices (debugging version)

`encoding_already_set`

SEE ALSO

The `gxFontPlatform`, `gxFontScript`, and `gxFontLanguage` data types are discussed in the chapter “Fonts Objects” of this book.

GXGetShapeEncoding

You can use the `GXGetShapeEncoding` function to determine the encoding set for the style object of a particular QuickDraw GX typographic shape.

```
gxFontPlatform GXGetShapeEncoding(gxShape source,
                                   gxFontScript *script,
                                   gxFontLanguage *language);
```

`source` A reference to the shape whose platform you want to determine.
`script` The script of the style object, returned by the function.
`language` The language of the style object, returned by the function.

function result The platform of the style object used by the specified shape.

Typographic Styles

DESCRIPTION

The `GXGetShapeEncoding` function returns the platform, script, and language of the style object associated with the shape object.

The function returns information about the platform, script, and language settings of the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleEncoding` (page 6-62) and `GXSetStyleEncoding` (page 6-62) functions to determine the platform, scripts, and language the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`style_is_nil`

SEE ALSO

The `gxFontPlatform`, `gxFontScript`, and `gxFontLanguage` data types are discussed in the chapter “Fonts Objects” of this book.

GXSetShapeEncoding

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

```
void GXSetShapeEncoding(gxShape target, gxFontPlatform platform,
                        gxFontScript script,
                        gxFontLanguage language);
```

<code>target</code>	A reference to the shape whose platform you want to alter.
<code>platform</code>	The platform of the style object. The <code>gxFontPlatform</code> data type is discussed in the chapter “Fonts Objects” of this book.
<code>script</code>	The script of the style object. The <code>gxFontScript</code> data type is discussed in the chapter “Fonts Objects” of this book.
<code>language</code>	The language of the style object. The <code>gxFontLanguage</code> data type is discussed in the chapter “Fonts Objects” of this book.

DESCRIPTION

The `GXSetShapeEncoding` function sets the style's platform, script, and language values for the shape named by the `target` parameter. The encoding specifies the way the font interprets the text stream into a series of character codes. A font may support one or more encodings.

If a style object is shared by more than one shape, `GXSetShapeEncoding` copies the style object so that only the shape in the `target` parameter is affected by the changes to the platform, script, and language values.

The function specifies the platform, script, and language settings for the style object associated with the shape object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleEncoding` (page 6-62) and `GXSetStyleEncoding` (page 6-62) functions to set the encoding of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES**Errors**

<code>shape_is_nil</code>	
<code>out_of_memory</code>	
<code>inconsistent_parameters</code>	(debugging version)

Notices (debugging version)

<code>style_platform_already_set</code>

SEE ALSO

The `gxFontPlatform`, `gxFontScript`, and `gxFontLanguage` data types are discussed in the chapter “Fonts Objects” of this book.

Getting and Setting the Text Attributes of a Style Object

The text attributes property of style objects specifies how QuickDraw GX alters glyph outlines or sets text to be horizontal or vertical. You use the `gxTextAttributes` data type, which is described on page 6-38, when retrieving or setting text attributes.

You can use the `GXGetStyleTextAttributes` function to retrieve the text attributes from a style object and the `GXSetStyleTextAttributes` function to specify the text attributes of a style object.

The `GXGetShapeTextAttributes` and `GXSetShapeTextAttributes` functions provide a way to retrieve and specify the text attributes for the style object associated with a particular shape.

GXGetStyleTextAttributes

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

```
gxTextAttribute GXGetStyleTextAttributes(gxStyle source);
```

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

function result The text attributes of the style object.

DESCRIPTION

The `GXGetStyleTextAttributes` function returns the text attributes of the style object specified by the `source` parameter.

ERRORS, WARNINGS, AND NOTICES

Errors

`shape_is_nil`
`out_of_memory`

SEE ALSO

See “Text Attributes” on page 6-38 for a listing of text attributes.

GXSetStyleTextAttributes

You can use the `GXSetStyleTextAttributes` function to set or change the text attributes of a style object.

```
void GXSetStyleTextAttributes(gxStyle target,  
                             gxTextAttribute attributes);
```

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

`attributes` The new text attributes for the style.

DESCRIPTION

The `GXSetStyleTextAttributes` function sets the text attributes of the style object specified by the `target` parameter to those specified in the `attributes` parameter.

You should always get the current settings of the text attributes before setting any of them. The `GXSetStyleTextAttributes` function replaces all of the attributes currently associated with the shape; if you want any attributes to remain the same, you must include them in the call, unless you want to set them all explicitly.

ERRORS, WARNINGS, AND NOTICES

Errors

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

Notices (debugging version)

`text_attributes_already_set`

SEE ALSO

Text attributes are described on page 6-38.

GXGetShapeTextAttributes

You can use the `GXGetShapeTextAttributes` function to determine which text attributes are set for the style object of a particular QuickDraw GX typographic shape.

```
gxTextAttribute GXGetShapeTextAttributes(gxShape source);
```

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

function result The text attributes of the style object attached to the *source* specified by shape.

DESCRIPTION

The `GXGetShapeTextAttributes` function returns the text attributes of the style object associated with the shape specified by the *source* parameter.

This function provides a convenient way to determine the text attributes of a shape without calling the `GXGetShapeStyle` function to obtain a reference to the shape's style object. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleTextAttribute` and `GXSetStyleTextAttribute` functions to determine the attributes of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

`shape_is_nil`
`out_of_memory`

SEE ALSO

See "Text Attributes" on page 6-38 for a listing of text attributes.

GXSetShapeTextAttributes

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

```
void GXSetShapeTextAttributes(gxShape target,
                             gxTextAttribute attributes);
```

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

`attributes` The new set of text attributes.

DESCRIPTION

The `GXSetShapeTextAttributes` function sets the text attributes of the style object associated with the shape specified by the `target` parameter. If a style object is shared among shapes, `GXSetShapeTextAttributes` copies the style object so that only the shape in the `target` parameter is affected by the changes to the text attributes.

This function provides a convenient way to set the style attributes of a shape without calling the `GXGetShapeStyle` function to obtain a reference to the shape's style object.

You can use this function in combination with the `GXGetShapeTextAttributes` function (page 6-67) to set or clear single style attributes. However, the glyph and layout shapes may have arrays of styles in their geometries and therefore do not necessarily use the style object attached to the shape object. In this case, you should use the `GXGetStyleTextAttribute` (page 6-66) and `GXSetStyleTextAttribute` (page 6-66) functions to set the attributes of the styles stored in the shape's geometry.

ERRORS, WARNINGS, AND NOTICES

Errors

`shape_is_nil`

`out_of_memory`

`parameter_out_of_range`

(debugging version)

Notices (debugging version)

`text_attributes_already_set`

SEE ALSO

Text attributes are described on page 6-38.

Summary of Typographic Styles

Constants and Data Types

```
enum gxTextAttributes {
    gxAutoAdvanceText      = 0x0001,
    gxNoContourGridText    = 0x0002,
    gxNoMetricsGridText    = 0x0004,
    gxAnchorPointsText     = 0x0008,
    gxVerticalText         = 0x0010,
    gxNoOpticalScaleText   = 0x0020
} ;

typedef long gxTextAttribute;

#define gxLeftJustify 0
#define gxCenterJustify (fract1/2)
#define gxRightJustify fract1
#define gxFillJustify -1

enum gxLayerFlags;{
    gxUnderlineAdvanceLayer = 0x0001,
    /* a gxLine is drawn through the advances */
    gxSkipWhiteSpaceLayer = 0x0002,
    /* except characters describing white space */
    gxUnderlineIntervalLayer = 0x0004,
    /* (+ gxStringLayer) a gxLine is drawn through the gaps
    between advances */
    gxUnderlineContinuationLayer = 0x0008,
    /* (+ gxStringLayer) join this underline with
    another face */
    gxWhiteLayer = 0x0010,
    /* the layer draws to white instead of black */
    gxClipLayer = 0x0020,
    /* the characters define a clip */
    gxStringLayer = 0x0040
    /* all characters in run are combined */
}

typedef long gxLayerFlag;
```

Typographic Styles

```

typedef struct {
    gxShapeFill outlineFill; /* outline framed or filled */
    gxLayerFlag flags;       /* various additional effects */
    gxStyle    outlineStyle; /* outline */
    gxTransform outlineTransform; /* italic, condense, extend */
    gxPoint    boldOutset; /* bold */
} gxFaceLayer;

typedef struct {
    long          faceLayers; /* layer to implement shadow */
    gxMapping      advanceMapping; /* algorithmic change to advance
                                   width */
    gxFaceLayer    gxFaceLayer[gxAnyNumber]; /* zero or more face
                                   layers describing the face */
} gxTextFace;

```

Functions

Getting and Setting the Font of a Style Object

```

gxFont GXGetStyleFont      (gxStyle source);
void GXSetStyleFont        (gxStyle target, gxFont aFont);
gxFont GXGetShapeFont      (gxShape source);
void GXSetShapeFont        (gxShape target, gxFont aFont);

```

Getting and Setting the Text Face

```

long GXGetStyleFace        (gxStyle source, gxTextFace *face);
void GXSetStyleFace        (gxStyle target, const gxTextFace *face);
long GXGetShapeFace        (gxShape source, gxTextFace *face);
void GXSetShapeFace        (gxShape target, const gxTextFace *face);

```

Getting and Setting the Text Size of a Style Object

```

Fixed GXGetStyleTextSize   (gxStyle source);
void GXSetStyleTextSize    (gxStyle target, Fixed size);
Fixed GXGetShapeTextSize   (gxShape source);
void GXSetShapeTextSize    (gxShape target, Fixed size);

```

Getting and Setting the Alignment of a Style Object

```

fract GXGetStyleJustification
                                   (gxStyle source);
void GXSetStyleJustification
                                   (gxStyle target, fract justify);

```

Typographic Styles

```

fract GXGetShapeJustification
    (gxShape source);
void GXSetShapeJustification (gxShape target, fract justify);

```

Getting and Setting the Style Font Variations of a Style Object

```

long GXGetStyleFontVariations
    (gxStyle source, gxFontVariation variations[]);
void GXSetStyleFontVariations
    (gxStyle target, long count,
     const gxFontVariation variations[]);
long GXGetShapeFontVariations
    (gxShape source, gxFontVariation variations[]);
void GXSetShapeFontVariations
    (gxShape target, long count,
     const gxFontVariation variations[]);

```

Retrieving the Elements in a Font Variation Suite

```

void GXGetStyleFontVariationSuite
    (gxStyle source, gxFontVariation variations[]);
void GXGetShapeFontVariationSuite
    (gxShape source, gxFontVariation variations[]);

```

Retrieving Font Metrics

```

void GXGetStyleFontMetrics (gxStyle sourceStyle, gxPoint* before,
                           gxPoint* after, gxPoint* caretAngle,
                           gxPoint* caretOffset);
void GXGetShapeFontMetrics (gxShape source, gxPoint* before, gxPoint* after,
                           gxPoint* caretAngle, gxPoint* caretOffset);
void GXGetShapeLocalFontMetrics
    (gxShape sourceShape, gxPoint* before,
     gxPoint* after, gxPoint* caretAngle,
     gxPoint* caretOffset);
void GXGetShapeDeviceFontMetrics
    (gxShape sourceShape, gxViewPort port,
     gxViewDevice device, gxPoint* before,
     gxPoint* after, gxPoint* caretAngle,
     gxPoint* caretOffset);

```

Getting and Setting the Encoding of a Style Object

```

gxFontPlatform GXGetStyleEncoding
                                (gxStyle source, gxFontScript *script,
                                 gxFontLanguage *language);

void GXSetStyleEncoding         (gxStyle target, gxFontPlatform platform,
                                 gxFontScript script, gxFontLanguage language);

gxFontPlatform GXGetShapeEncoding
                                (gxShape source, gxFontScript *script,
                                 gxFontLanguage *language);

void GXSetShapeEncoding        (gxShape target, gxFontPlatform platform,
                                 gxFontScript script, gxFontLanguage language);

```

Getting and Setting the Text Attributes of a Style Object

```

gxTextAttribute GXGetStyleTextAttributes
                                (gxStyle source);

void GXSetStyleTextAttributes   (gxStyle target, gxTextAttribute attributes);

gxTextAttribute GXGetShapeTextAttributes
                                (gxShape source);

void GXSetShapeTextAttributes   (gxShape target, gxTextAttribute attributes);

```