

This chapter introduces the text-handling capabilities of QuickDraw GX and defines important typographic terms related to text. If you are developing a QuickDraw GX application that uses text, read this chapter before reading any other parts of this book.

This chapter assumes that you have read the book *Inside Macintosh: QuickDraw GX Objects* and that you know what shape objects and style objects are.

This chapter starts by outlining the different components that make up text. It then describes

- how text is measured and stored
- how you can arrange text on a display device
- how you can adjust the text in various ways by affecting the text direction, kerning, alignment, justification, and line breaks
- how your application can draw, highlight, and hit-test text

Typography and QuickDraw GX

Text has traditionally been defined as the written representation of spoken language. QuickDraw GX extends this definition by treating text as both text and graphics and by allowing you to use special typographic features to generate and manipulate fully editable, text-related shapes.

Because each line of text is a QuickDraw GX shape, you can modify it as you would any other graphic shape, yet the shape still maintains its identity and editability as a text line. With QuickDraw GX, you can use one or more of the typographic shapes described in this book for simple word-processing tasks as well as for laying out more complex, typographically sophisticated text lines.

Typographic shapes have the same fundamental structure as other shapes. The geometry of a typographic shape contains its characters, just as a rectangle shape's geometry contains the points that make up the upper-left and lower-right corners of the rectangle. There are three kinds of typographic shapes, each with specific characteristics and properties:

- A **text shape** consists of a string of one or more characters or glyphs, all to be displayed in the same font with the same typestyle.
- A **glyph shape** consists of one or more characters or glyphs, each of which can be independently located, rotated, sized, and styled.
- A **layout shape** consists of a line of text that may be in multiple languages, and which may be displayed with multiple writing directions (including vertical), with ligatures and other contextual forms, and with other sophisticated formatting and stylistic properties.

The three shapes are described as a group in the chapter "Typographic Shapes" and individually in the chapters "Text Shapes," "Glyph Shapes," and "Layout Shapes."

Characters, Glyphs, and Fonts

A writing system's alphabet, numbers, punctuation, and other writing marks consist of characters. A **character** is a symbolic representation of an element of a writing system; it is the concept of, for example, "lowercase a" or "the number 3." It is an abstract object, defined by custom in its own language.

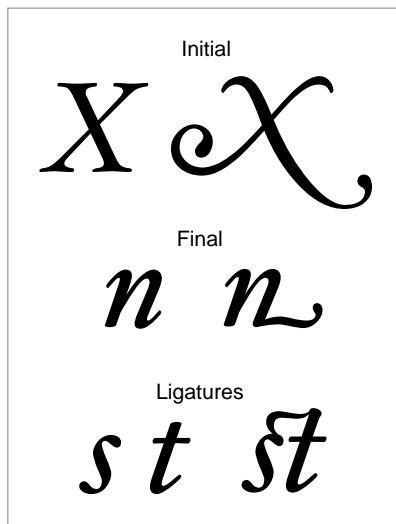
As soon as you write a character, however, it is no longer abstract but concrete. The exact shape by which a character is represented is called a **glyph**. The "characters" that QuickDraw GX places on the screen are really glyphs.

Glyphs and characters do not necessarily have a one-to-one correspondence. For example, a single character may be represented by one or more glyphs (the character "lowercase i" could be represented by the combination of glyphs "i" and "."), and a single glyph can represent two or more characters (the single glyph "fi" could represent the two characters "f" and "i").

Context—where the glyph appears in a line of text—also affects which glyph represents a character. Figure 1-1 shows examples of such contextual forms in a Roman font. Different forms of a glyph are used according to whether the glyph stands alone or occurs at the beginning of a word, occurs at the end of a word, or forms part of a new glyph, as in a ligature.

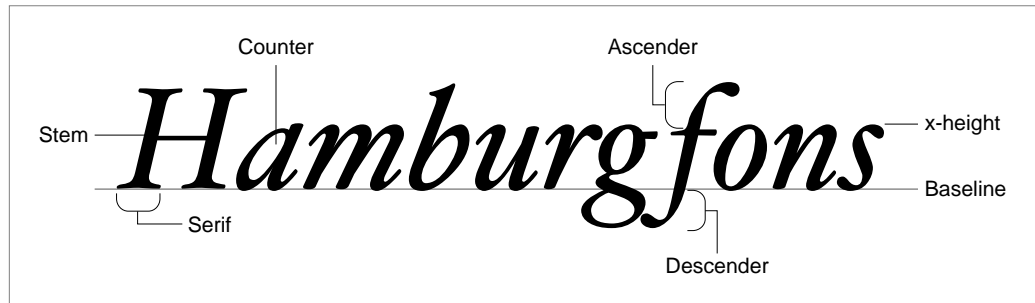
A **font** is a collection of glyphs, all of similar design, that constitute one way to represent the characters of one or more language.

Figure 1-1 Contextual forms in a Roman font



Fonts usually have some element of design consistency, such as the shape of the ovals (known as the **counter**), the design of the stem, stroke thickness, or the use of **serifs**, which are the fine lines stemming from the upper and lower ends of the main strokes of a letter. Figure 1-2 shows some of the elements of glyphs that indicate they are members of the same family.

Figure 1-2 Elements that distinguish glyphs of a Roman font



A font always has a full name—for example, Geneva Regular or Times Bold. The full name determines which family the font belongs to and what typestyle it represents. (Typestyles are discussed on page 1-10.) The font Geneva Italic, for example, shares many characteristics with Geneva Regular, but all of the glyphs slant at a certain angle. Though different, these fonts are part of the same font family. A **font family** is a group of fonts that share certain characteristics and have a common family name. Each font family has its own name, such as “New York,” “Geneva,” or “Symbol.” Several fonts may have the same family names (such as Geneva, Geneva Bold, and so on) but are stored separately—these fonts are still part of the same font family.

Note

QuickDraw GX does not use the 'FOND' resource to determine what fonts are part of which font family. It uses information in the naming table of each font—a table that every QuickDraw GX font has. ♦

Encodings

For Roman fonts that have a one-to-one correspondence between glyphs and characters, an application can access the proper glyph using 1-byte **character codes**. As Figure 1-3 shows, Macintosh Roman character codes are hexadecimal numbers from \$00 through \$FF that represent the characters corresponding to a key or key combination. The figure shows 1-byte character codes; but 2-byte character codes are also used—for example, in Asian fonts, which may have 8,000 or more glyphs.

Figure 1-3 The Standard Roman character set

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	nul	dle	sp	0	@	P	`	p	À	ê	†	∞	¿	-	‡	🍏
x1	soh	DC1	!	1	A	Q	a	q	Å	ë	°	±	ı	—	·	Ò
x2	stx	DC2	"	2	B	R	b	r	Ç	ì	¢	≤	¬	“	,	Ó
x3	etx	DC3	#	3	C	S	c	s	É	í	£	≥	√	”	„	Ô
x4	eot	DC4	\$	4	D	T	d	t	Ñ	î	§	¥	ƒ	‘	‰	Ù
x5	enq	nak	%	5	E	U	e	u	Ö	ï	•	μ	≈	’	Â	ı
x6	ack	syn	&	6	F	V	f	v	Ü	ñ	¶	∂	Δ	÷	Ê	^
x7	bel	etb	'	7	G	W	g	w	á	ó	ß	Σ	<<	◊	Á	~
x8	bs	can	(8	H	X	h	x	à	ò	®	Π	>>	ÿ	Ë	-
x9	ht	em)	9	I	Y	i	y	â	ô	©	π	...	ÿ	È	˘
xA	lf	sub	*	:	J	Z	j	z	ä	ö	™	ƒ	nbsp	/	í	.
xB	vt	esc	+	;	K	[k	{	ã	õ	'	a	À	¤	Î	°
xC	ff	fs	,	<	L	\	l		å	ú	"	°	Ã	<	Ï	¸
xD	cr	gs	-	=	M]	m	}	ç	ù	≠	Ω	Ö	>	ì	”
xE	so	rs	.	>	N	^	n	~	é	û	Æ	æ	Œ	fi	Ó	¸
xF	si	us	/	?	O	_	o	del	è	ü	Ø	ø	œ	fl	Ô	˘

Fonts also associate each glyph with a 2-byte code called its **glyph code**. Different fonts may have different glyph codes for the same glyphs, and a single font may have several glyph codes associated with a particular character because several glyphs may represent that character. Because the font and general textual context determine which glyph and which glyph codes represent characters, QuickDraw GX transparently handles the details of mapping character codes to the correct glyph codes. Your application does not have to handle the details of obtaining glyphs from a font.

Note

Your application will usually deal with character codes, since those correspond most closely with what the user types. However, QuickDraw GX permits you to deal with glyph codes, if that is more appropriate to your application's functionality and needs. ♦

Different languages may have different requirements in terms of which glyphs they want from a font. A font contains some number of **character encodings**. Each encoding is an internal conversion table for interpreting a specific character set—that is, a way to map a character code to glyph code for that font.

The reason a font can have multiple encodings is that the requirements for each writing system that the font supports may be different. A **writing system** is a method of depicting words visually. It consists of a character set and a set of rules for displaying, ordering, and formatting the glyphs associated with those characters. Writing systems can differ in line direction, the direction in which their glyphs are read; the size of the character set used to represent the script; and contextual variation (that is, whether a glyph changes according to its position relative to other glyphs). Writing systems have specific requirements for text display, text editing, character set, and fonts. A writing system—for instance, the Roman system—can serve one or several languages, such as French, Italian, and Spanish.

Text Storage

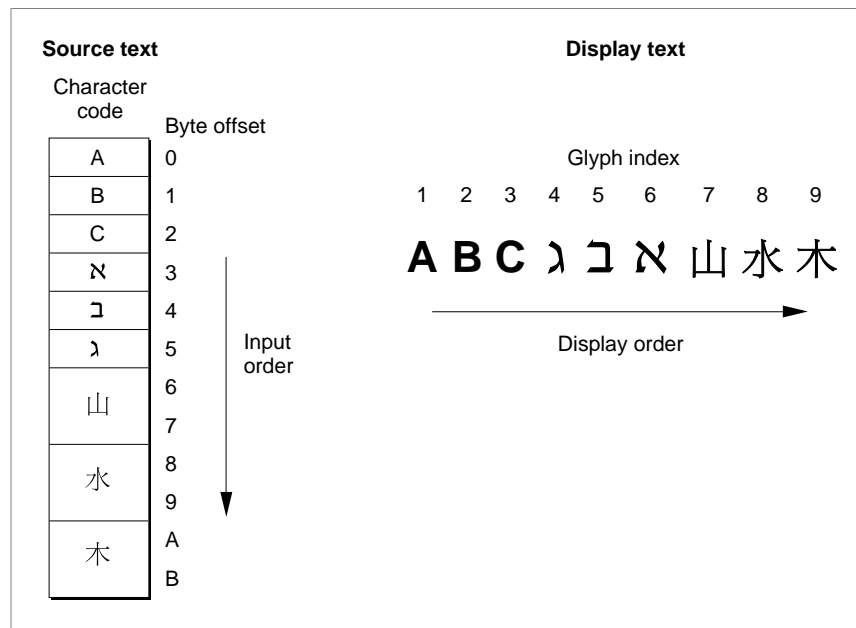
QuickDraw GX stores text in a typographic shape's geometry as a sequence of character codes or glyph codes. The **storage order** is the order in which text is stored. A shape may contain 1-byte or 2-byte codes, or a mixture of both. Using information in the style object, QuickDraw GX determines whether a character code is 1 or 2 bytes. The text stored in a shape is the **source text**; the text displayed is the **display text**, as shown in Figure 1-4.

Display order is the left-to-right (or top-to-bottom) order in which glyphs are drawn. For text shapes, and by default for glyph shapes, storage order and display order are the same. For layout shapes, however, QuickDraw GX expects you to store your text in **input order**, which is the “logical” order, or the order in which the characters, *not* glyphs, would be read or pronounced in the language of the text. Because text of different languages may be read from left to right, right to left, or top to bottom, the input order is not necessarily the same as the display order of the text when it is drawn. Your application needs to differentiate between the order in which the character codes are stored in the shape and the order in which the corresponding glyphs are displayed.

Figure 1-4 shows Hebrew glyphs that are stored one way and displayed another way in a layout shape.

Note

In Figure 1-4 and throughout this book, text in computer memory is drawn as a vertical table of codes, representing sequential (downward) storage of text characters in a buffer. Some diagrams also include byte offsets in the buffer, and even miniature representations of the characters themselves in a given language. ♦

Figure 1-4 Input order and display order

As shown in Figure 1-4, the character codes that make up the text are numbered using zero-based **offsets**. Therefore, the first character code in the figure has an offset of 0.

QuickDraw GX uses a different numbering scheme to index the glyphs that are actually displayed. The **glyph index**, which gives the glyph's position in the display order, always starts at 1. Therefore, the offset of the character code and the index of the corresponding glyph may be different. Also, each glyph has a single index, even if its character code is 2 bytes long (as in the case of Chinese characters). In Figure 1-4, the character code offset of the uppercase "A" is 0, but the glyph's index is 1. Likewise, the last Chinese glyph begins at character code offset A but has an index of 9.

For layout shapes, QuickDraw GX provides functions to map back and forth from byte offsets to glyph indexes. For text and glyph shapes, such mapping is not needed because with those shapes a character offset corresponds one-to-one with a glyph index.

Text Measurements

Most users use point size to specify the size of the glyphs in a document. **Point size** indicates the size of a font's glyphs as measured from the baseline of one line of text to the baseline of the next line of single-spaced text; in the United States, point size is measured in **typographic points**, and there are 72.27 points per inch. However, QuickDraw GX and the PostScript™ language both define 1 point to be exactly $1/72$ of an inch. QuickDraw GX permits fractional point sizes.

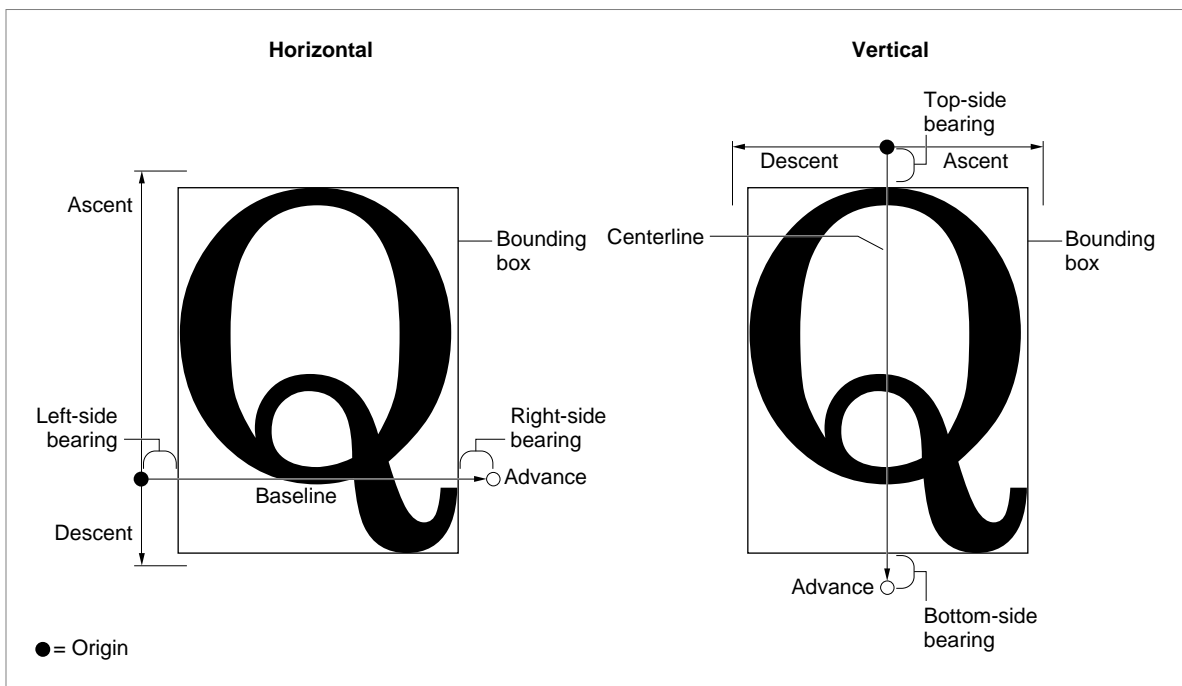
Although point size is a useful measure of the size of text, you may wish to use more exact measurements for greater control over placement of the glyphs on the display device.

Font designers use a special vocabulary for the measurements of different parts of a glyph. Figure 1-5 shows the terms describing the most frequently used measurements.

The **bounding box** of a glyph is the smallest rectangle that entirely encloses the drawn parts of the glyph. The **glyph origin** is the point that QuickDraw GX uses to position the glyph when drawing. In Figure 1-5, notice that there is some space between the glyph origin and the edge of the bounding box: this space is the glyph's **left-side bearing**. The left-side bearing value can be negative, which decreases the space between adjacent characters. The **right-side bearing** is space on the right side of the glyph; this value may or may not be equal to the value of the left-side bearing. The **advance width** is the full horizontal width of the glyph as measured from its origin to the origin of the next glyph on the line, including the side bearings on both sides.

Most glyphs in Roman fonts appear to sit astride the **baseline**, an imaginary horizontal line. The **ascent** is a distance above the baseline, chosen by the font's designer and the same for all glyphs in a font, that often corresponds approximately to the tops of the uppercase letters in a Roman font. Uppercase letters are chosen because, among the regularly used glyphs in a font, they are generally the tallest. The **descent** is a distance below the baseline that usually corresponds to the bottoms of the descenders (the "tails" on glyphs such as "p" or "g"). The descent line is the same distance from the baseline for all glyphs in the font, whether or not they have descenders. The sum of ascent plus descent marks the **line height** of a font.

Figure 1-5 Terms for glyph measurements



For vertical text, font designers may use additional measurements. The **top-side bearing** is the space between the top of the glyph and the top edge of the bounding box. The **bottom-side bearing** is the distance from the bottom of the bounding box to the origin of the next glyph. For vertical text, the **advance height** is the sum of the top-side bearing, the bounding-box height, and the bottom-side bearing.

These metrics are useful if, for example, you want to display a horizontal font vertically. Likewise, vertical fonts such as Kanji may also have horizontal metrics.

Typestyles

Glyphs can be differentiated not only by font but by typestyle. 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. Some of the typical typestyles available on the Macintosh computer include plain, bold, italic, underline, outline, shadow, condensed, and extended. Other styles that may be available are Demibold, Extra Condensed, or Antique.

Font Variations and Instances

A **font variation** is a setting along a particular variation axis. Font variations allow your application to produce a range of typestyles algorithmically.

Each **variation axis** has a name identifying the typestyle that the axis represents (such as weight or width), a tag to represent that name (such as 'wght'), a set of maximum and minimum values for the axis, and the default value of the axis. The weight axis, for example, governs the possible values for the weight of the font; the minimum value may produce the lightest appearance of that font, the maximum value the boldest. The default value is the position along the variation axis value at which that font falls normally.

Because the axis is created by the font designer, font variations can be optimized for their particular font. Figure 1-6 shows a range of possible weights for a glyph, from the minimum weight to the maximum weight.

A **font instance** is a set of named variations identified by the font designer that matches specific values along the available variation axes and associates those values with a name. For example, suppose a font has the variation axis 'wght' with a minimum value of 0.0, a default of 0.5, and a maximum value of 1.0. The corresponding font instance might have the name "Demibold" with a value along that variation axis of 0.8.

Figure 1-6 Font variations along a variation axis



In Figure 1-6, the variation axis value of the glyph at the far right could represent the named instance “Extra Bold,” whereas the glyph at the far left could represent the named instance “Light.” The other values represented in the figure could likewise have instance names.

Font variations and font instances give your application the ability to provide whatever typesyles the font designer has decided to include with the font.

Text Faces

If the desired typestyle is not available as a separate font and cannot be produced as a font variation, your application can generate a **text face**, which is an algorithmic way of producing typesyles. You can use text faces to produce bold, italic, condensed, and other typical Macintosh typesyles. You can also use them to create unusual typesyles not supported anywhere else; for example, you can create a “sunburst” text face, as shown in Figure 1-7.

Figure 1-7 An example of an unusual text face



Laying Out Text

When you arrange text on a display device, QuickDraw GX offers a simple approach: you can put the information into one of the typographic shapes and then display it. Depending on the shape type, QuickDraw GX automatically handles many aspects of the text display. However, QuickDraw GX also offers you the ability to adjust the text in various ways, to gain greater control over the presentation and arrangement of the text. For example, you can affect

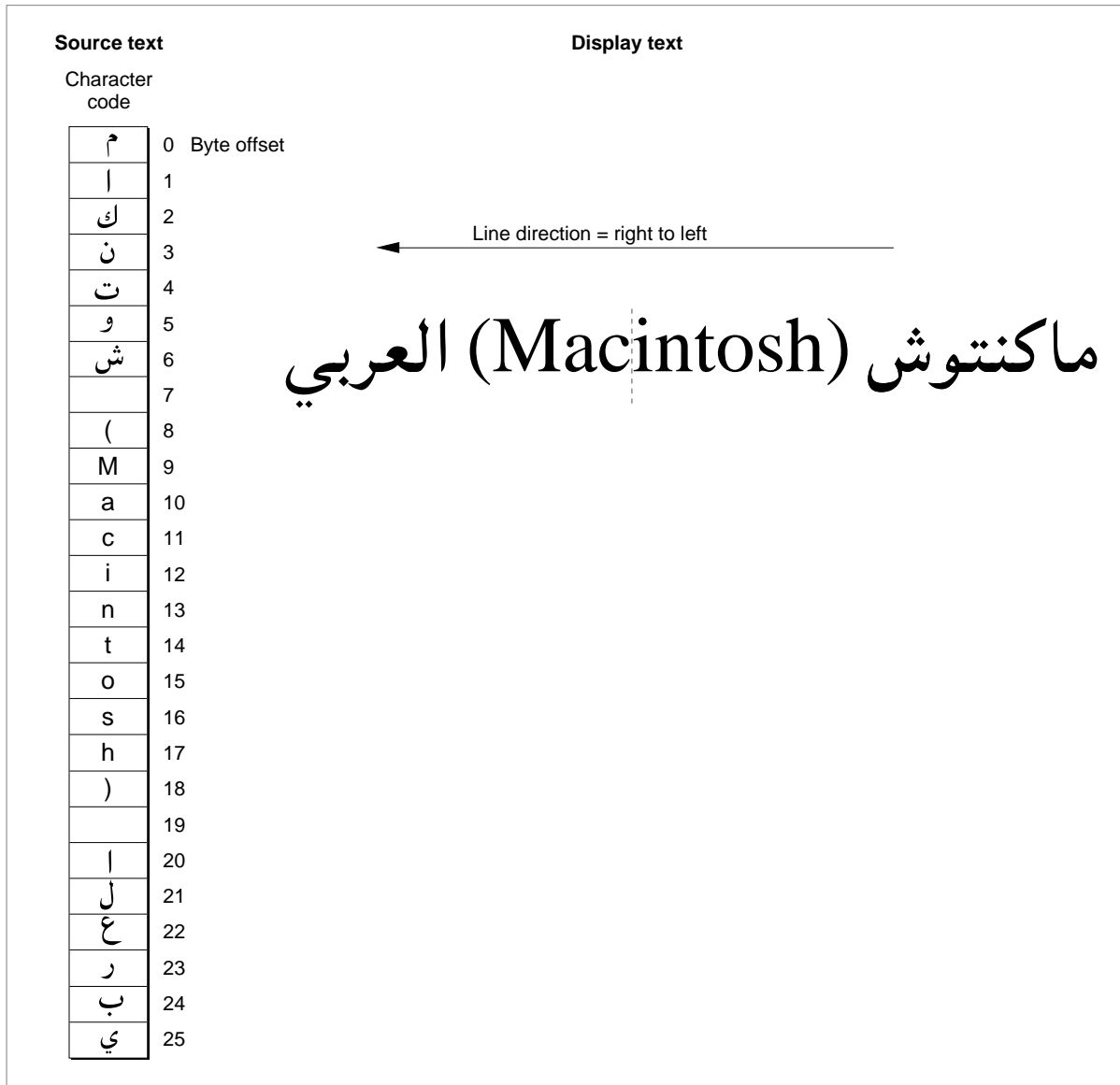
- text direction and baselines
- text runs, style runs, and direction runs
- contextual forms and ligatures
- alignment and justification
- kerning and tracking
- line breaks

This section describes some of the general concepts governing how you can affect the text presented by your application. Actual implementation of these concepts is described in various chapters in this book.

Text Direction and Baselines

Text direction consists of text orientation (horizontal or vertical) and the direction in which the text is read. Text in your application can be oriented in three common directions: horizontally, left to right; horizontally, right to left; and vertically, top to bottom. QuickDraw GX allows your application, for example, to draw lines of text in multiple directions, as shown in Figure 1-8.

Figure 1-8 Multi-direction text



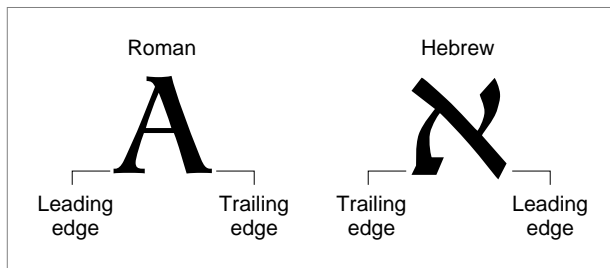
The text in Figure 1-8 contains a mixture of English and Arabic, with the primary direction being right to left. The figure also shows the character codes and byte offsets of the source text.

Leading Edges and Trailing Edges

Because text has a direction, the concept of which glyph comes “first” in a line of text cannot always be limited to the visual terms “left” and “right.” The **leading edge** is defined as the edge of a glyph you first encounter—such as the left foot of a Roman glyph—when you first read the text that includes that glyph. The **trailing edge** is the edge of a glyph encountered last.

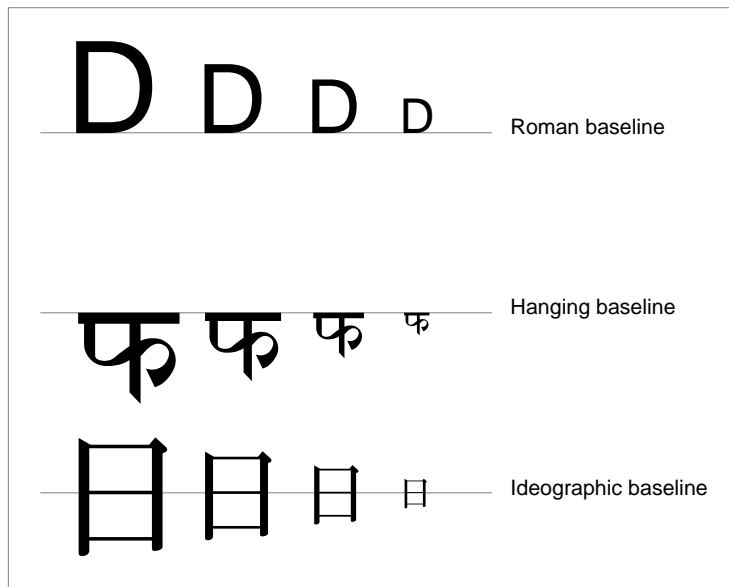
Figure 1-9 shows how the concepts of leading edge and trailing edge change depending on the characteristics of the glyph. In the first example—a Roman glyph—the leading edge is on the left, because the reader encounters that side first. In the second example, the leading edge of the Hebrew glyph is on the right for the same reason. For more information, see the chapter “Layout Carets, Highlighting, and Hit-testing” in this book.

Figure 1-9 Leading edges and trailing edges



Baselines

A **baseline** is an imaginary line that coincides with some point in a font—for example, the bottom, middle, or top of each glyph. The baseline of a glyph defines the position of the glyph with respect to other glyphs at different point sizes when all the glyphs are aligned. It represents a stable platform from which glyphs of different sizes and different writing systems grow proportionally, as shown in Figure 1-10.

Figure 1-10 Baselines for different sizes of a glyph and for different writing systems

Note that, depending on the writing system, the baseline may be above, below, or through the center of each glyph.

QuickDraw GX provides your application with capabilities for using multiple baselines. For more information, see the chapter “Layout Line Control” in this book.

Various baselines can also be used to create special effects, such as drop capitals. (A **drop capital** is an initial capital letter that is much larger than surrounding glyphs and embedded in them.) Figure 1-11 is an example of drop capitals formed solely on the basis of the baselines in the font. The default baseline for this text is the Roman baseline for 18-point type. The hanging baseline of the drop capitals aligns with the hanging baseline of the regular text, creating the effect shown.

Figure 1-11 Drop capitals

Text Runs, Style Runs, and Direction Runs

In any segment of contiguous text, certain parts stand out as belonging together, because the glyphs share a certain font, typestyle, or direction. For the purposes of referring to individual segments of text, you can think of sequences of glyphs that are contiguous in memory and share a set of common attributes as **runs**.

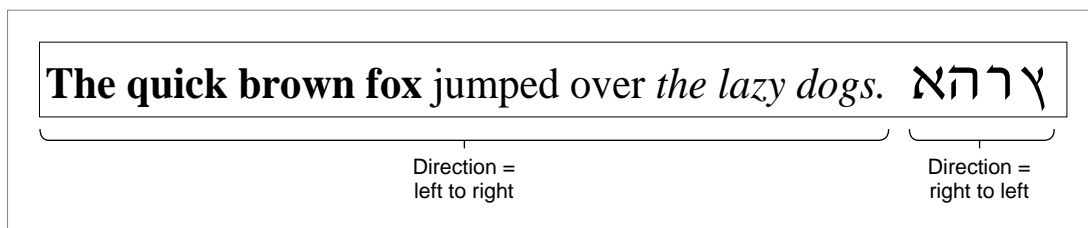
The complete text in a shape is one **text run**. (A shape has only one text run internally, although for layout shapes your application can provide multiple text runs.) A sequence of glyphs continuous in memory that share the same style object is a **style run**. As Figure 1-12 shows, a text run can be subdivided into several style runs.

Figure 1-12 Three style runs in a line of text



A sequence of contiguous glyphs that share the same text direction is a **direction run**. As with text runs and style runs, the number of direction runs does not necessarily correlate to the number of style runs available, as shown in Figure 1-13.

Figure 1-13 Two direction runs in a line of text



For more information on using text runs, style runs, and direction runs, see the chapters “Layout Shapes” and “Glyph Shapes” in this book.

Contextual Forms and Ligatures

A glyph's position next to other glyphs or its position in a word or a line of text may determine its appearance. For some writing systems, such as Roman, alternate glyphs are used for aesthetic reasons; in other writing systems, use of alternate forms is required.

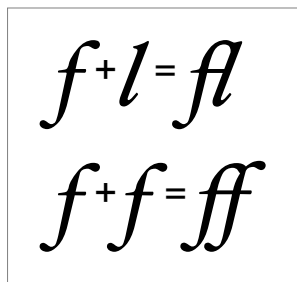
A **contextual form** is an alternate form of a glyph that is chosen depending on the glyph's placement in a certain context, such as a certain word or line. Some contextual forms of initial and final forms of glyphs in a Roman font are shown in Figure 1-1 on page 1-4. Other writing systems, such as Arabic, require different contextual forms of glyphs according to where they appear. Figure 1-14 shows the forms of the Arabic letter "ha" that appear alone and at the beginning, middle, or end of a word. The same character code is used for each case; QuickDraw GX finds the appropriate glyph code.

Figure 1-14 Contextual forms of the Arabic letter "ha"

Independent	Final	Medial	Initial
ه	ـه	هـ	هـ

Ligatures are two or more glyphs combined to form a single new glyph (whereas contextual forms are variations on the shape of one glyph). In the Roman writing system, ligatures are generally an optional aesthetic refinement; in other writing systems, special ligatures are required when certain glyphs appear next to one another. Some examples of ligatures used in a Roman font are shown in Figure 1-15.

Figure 1-15 Examples of Roman ligatures



In general, the font contains all of the information needed to determine when your application should use the appropriate contextual forms and ligatures. If your application allows alternate forms of glyphs to be used, QuickDraw GX does the substitution for you.

Only the layout shape substitutes ligatures and contextual forms for the character codes stored in the shape when the shape is displayed, if you request that behavior. The text and glyph shapes do not perform contextual substitutions.

Alignment and Justification

Once you have the text, you can arrange it in the text area. The **text area** is the space on the display device in which the text should fit. The left, right, top, and bottom sides of that area are the **margins**.

How you arrange the text depends on the effect you want to achieve. There are two primary methods of arranging text: alignment and justification.

Alignment is the process of placing text in relation to one or both margins. You can set the alignment in the style object for glyph and text shapes, but not for layout shapes, which use a different mechanism to align lines of text.

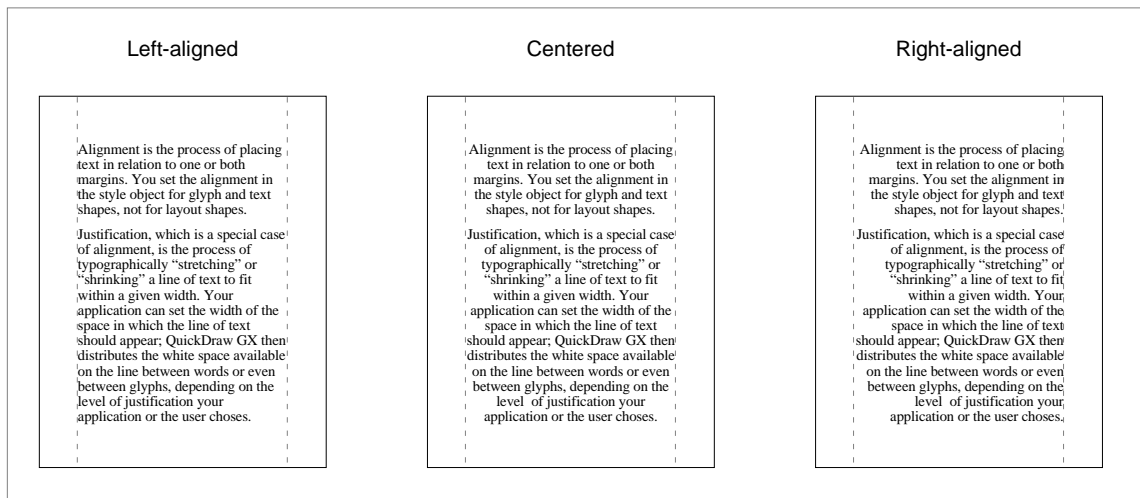
Figure 1-16 shows left, right, and center alignment of text.

Justification is the process of typographically “stretching” or “shrinking” a line of text to fit within a given width. Your application can set the width of the space in which the line of text should appear; QuickDraw GX then distributes the white space available on the line between words or even between glyphs, depending on the level of justification chosen.

For the layout shape, there are other means of aligning or justifying a line—for example, stretching a glyph or decomposing a ligature. QuickDraw GX can also handle complex justification such as that used in Arabic writing systems.

For more information about justification, see the chapter “Layout Line Control” in this book.

Figure 1-16 Different kinds of alignment



Kerning and Tracking

Kerning is an adjustment to the normal spacing between two or more specific glyphs. A **kerning pair** consists of two adjacent glyphs such that the position of the second glyph is changed with respect to the first. The font designer determines which glyphs participate in kerning and in what context. Any adjustments to glyph positions are specified relative to the point size of the glyphs. Kerning usually improves the apparent letter-spacing between glyphs that “fit together” naturally.

Figure 1-17 shows how glyphs are positioned differently with and without kerning. Note that the phrase is shorter when kerning is applied than when not.

Figure 1-17 Glyphs with and without kerning



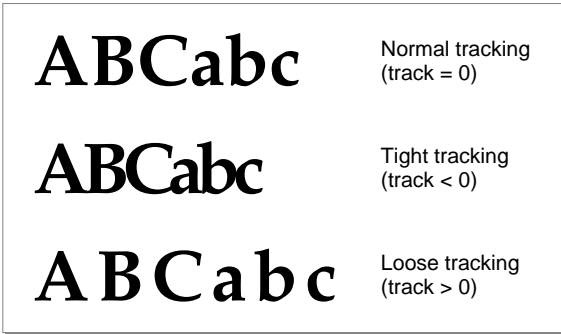
Cross-stream kerning allows the automatic movement of glyphs perpendicular to the line orientation of the text. (For example, when QuickDraw GX applies cross-stream kerning to horizontal text, the automatic movement is vertical; this feature is required for writing systems such as Taliq, which is used in the Urdu language.)

When your application lays out text, it has the option of using the interglyph spacing specified by the font designer or altering the spacing slightly in order to achieve a tighter fit between letters and improve the look of a line of text.

Your application can also use tracking. In **tracking**, space is adjusted between all glyphs in the run. You can increase or decrease interglyph spacing by using a **track setting**, which is a value that specifies the relative tightness or looseness of interglyph spacing. Positive track settings result in an increase in the looseness of all glyphs in the run. Negative track settings result in an increase in the tightness of all glyphs. Normal tracking, tight tracking, and loose tracking are shown in Figure 1-18.

For more information about kerning and tracking, see the chapter “Layout Styles” in this book.

Figure 1-18 Normal, tight, and loose tracking by the selection of track setting



Special Font Features

Some special features are available in certain fonts. You can increase the control a user has over the presentation of text in a document if you provide access to these features when they are available in a font; the font provides the functionality for using these features. Table 1-1 shows some of the currently defined features.

Table 1-1 Some special font features for layout shapes

Feature	Description
Ligatures	Permits selection from different ranges of ligatures.
Cursive connections	Controls the level of cursive connection in the font. This feature is used in fonts, such as cursive Roman fonts or Arabic fonts, in which glyphs are connected to each other.
Vertical substitution	Specifies that glyphs need to change their appearance in vertical runs of text.
Smart swashes	Controls contextual swash substitution, such as substituting a final glyph when a particular glyph appears as the end of a word.
Vertical position	Controls superscripts, subscripts, and ordinal forms.
Fractions	Governs selection and generation of fractions.
Overlapping glyphs	Prevents the collision of long tails on glyphs with the descenders of other glyphs.
Typographic extras	Allows fine typographic effects, such as the automatic conversion of two adjacent hyphens to an em dash.
Ornament sets	Governs nonletter ornament sets of glyphs.
Style options	Allows the font designer to group together collections of noncontextual substitutions into named sets.
Character shape	Specifies the use, with Chinese fonts, of the traditional or simplified character forms.

Some of these features, such as typographic extras, are fancy elements that provide the user with alternate forms of glyphs or other ornamental designs. Other features are contextual and absolutely necessary for using that font correctly—for example, cursive connectors for Arabic. (The fonts that require these features include them.)

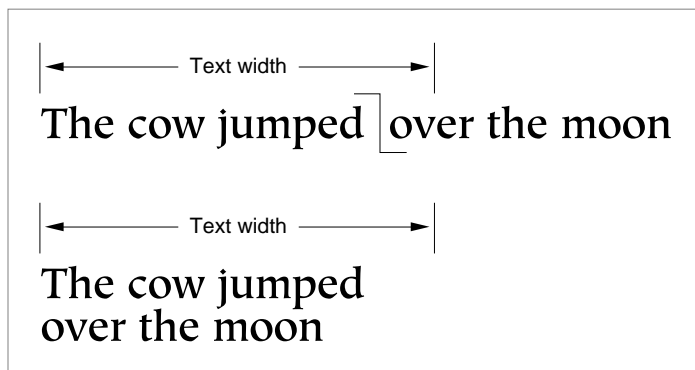
Only the layout shape allows you to take advantage of special typographic features in a font. For more information, see the chapters “Layout Shapes” and “Layout Styles” in this book.

Line Breaking

A typographic shape is designed to represent one line of text. Your application determines how wide the text area for a line should be. At times, a user enters a line of text that does not fit neatly in the given text area and overlaps one of the margins. When this happens, you break the line of text and wrap the text onto the next line.

Figure 1-19 shows a line break made on the basis of a simple algorithm: the application backs up in the source text of the shape to the trailing edge of the last white space and then carries over all of the text following that white space to the next line.

Figure 1-19 Determining where to break a line



Your application can devise more complex algorithms, such as breaking a word at an appropriate hyphenation point, if possible.

QuickDraw GX leaves the final decision about where to break the line up to your application. However, when you use the layout shape, QuickDraw GX provides you with a set of functions designed to help you determine the best place to break a line. For more information, see the chapter “Layout Line Control” in this book.

Drawing, Highlighting, and Hit-Testing Text

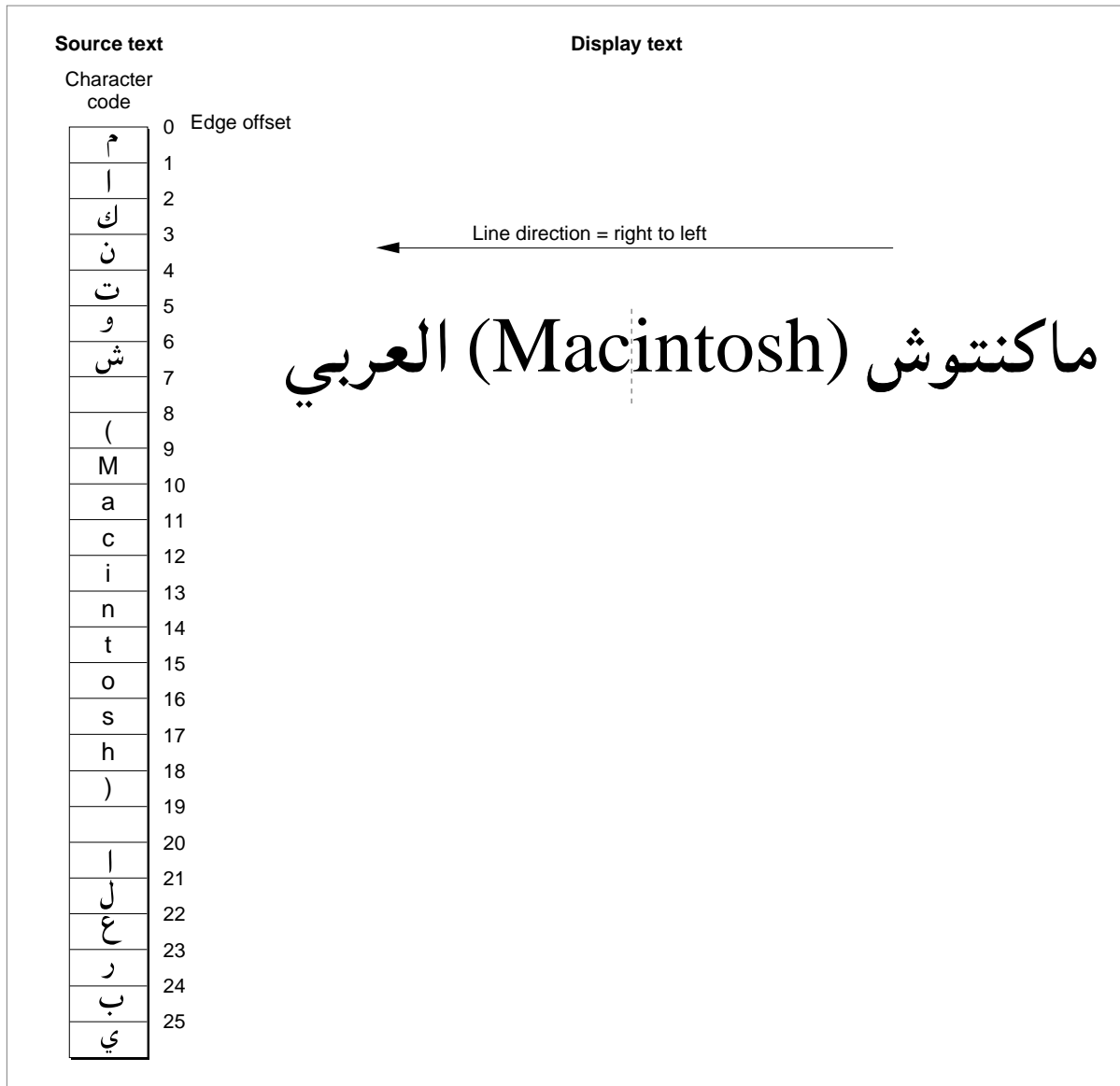
After you have laid out the text exactly as you want, you can draw it. If you want the user to interact with the text, you also need to support carets, highlighting, and hit-testing.

Drawing text using QuickDraw GX is simple. You draw the typographic shapes exactly as you might any other type of shape. There are no special requirements for drawing text; a single call, `GXDrawShape`, is all you need.

Carets

A **caret** is a single line that appears at the position in the text where the user can insert the next character. Carets indicate where the user can next add text. Figure 1-20 shows how a caret appears between glyphs of a word; if the user were to add new text at this point, the corresponding glyphs would appear between the “c” and the “i” of the word.

Remember that the glyphs in a line of text are numbered using a 1-based indexing scheme. (See “Text Storage” on page 1-7.) However, when you place a caret between glyphs, you need to be able to relate it to the insertion point: a point between byte offsets in the source text. The **edge offset** is a byte offset between character codes in the source text that corresponds to the caret location between glyphs. In Figure 1-20, the edge offset is 12; the glyphs on either side have indexes of 12 and 13.

Figure 1-20 Caret position

Highlighting

Highlighting is the display of text in inverse video or with a colored background. Figure 1-21 shows highlighting in some Arabic text.

Figure 1-21 Highlighting



The characters in memory corresponding to the glyphs that are highlighted make up the **selection range**, which indicates where the next editing operation is to occur. The characters in a selection range are always contiguous in memory, but their corresponding glyphs are not necessarily so onscreen.

If the selection range crosses a **direction boundary**—a point at which the direction of the displayed text changes—you may get **discontiguous highlighting**, as shown in Figure 1-22.

Figure 1-22 Discontiguous highlighting



QuickDraw GX provides functions that create carets, contiguous highlighting, and discontiguous highlighting for a layout shape. If you use one of the other types of shapes, you must create your own carets and highlighting. For more information, see the chapter “Layout Carets, Highlighting, and Hit-testing” in this book.

Hit-Testing

Caret handling and highlighting require you to convert from the edge offset to the screen position. You must also be able to convert from screen position to edge offset. For example, if the user clicks the mouse button while the cursor is in displayed text, you need to determine the offset in your text buffer equivalent to that mouse-down event. You can then use that information to set the insertion point or selection range.

Introduction to QuickDraw GX Typography

QuickDraw GX does most of the work of **hit-testing**, or converting a location within the line into the corresponding edge offset that corresponds to that location in the original string. Figure 1-23 shows a simple example of hit-testing. The user clicks on a glyph near its boundary with another glyph. QuickDraw GX translates the location of this mouse click into a point in the view port and to an offset in the source text that marks the equivalent boundary between character codes. It takes into account the glyph edge (the leading edge or trailing edge) nearest to which the click occurred and the most appropriate place to display the caret.

Figure 1-23 Hit-testing



QuickDraw GX provides functions that perform hit-testing on any type of shape, and these functions can be used on any typographic shape. In general, if you are using layout shapes, you should use the layout hit-testing function, which provides more information about complex situations, such as when the shape contains multi-directional text (for example, English and Hebrew).

For more information, see the chapter “Layout Carets, Highlighting, and Hit-Testing” in this book.