This chapter describes the format of the compressed data stream that results when the QuickDraw GX `GXFlattenShape` function is used. It also describes the use of such data streams by print files and portable digital documents (PDDs). Read this chapter if you need to uncompress QuickDraw GX stream format data and cannot use the QuickDraw GX `GXUnflattenShape` function.

Before reading this chapter, you should be familiar with the information in the chapters "Introduction to QuickDraw GX Objects" and "Shape Objects" in *Inside Macintosh: QuickDraw GX Objects*.

The `GXFlattenShape` and `GXUnflattenShape` functions and additional information about the objects contained in the data stream are described in *Inside Macintosh: QuickDraw GX Objects*. For more information on graphic shapes, see the shape-specific chapters in *Inside Macintosh: QuickDraw GX Graphics*. For more information on typographic shapes, see the shape-specific chapters in *Inside Macintosh: QuickDraw GX Typography*. For more information on print files and portable digital documents, see the chapter "Advanced Printing Features" of *Inside Macintosh: QuickDraw GX Printing*.

This chapter first describes the QuickDraw GX stream format, print file organization, and portable digital documents. It then shows how you can

- use the GraphicsBug utility to flatten QuickDraw GX shapes
- analyze flattened shape data streams
- obtain information from a print file

About QuickDraw GX Stream Format

A QuickDraw GX **data stream** is a highly structured sequence of bytes that contains all of the information required to store, print, or display QuickDraw GX objects.

QuickDraw GX provides a simple method for creating and interpreting a QuickDraw GX data stream for shape objects. The `GXFlattenShape` function creates the data stream and the `GXUnflattenShape` function reconstructs objects from the data stream that the `GXFlattenShape` function previously created.

When the `GXFlattenShape` function converts shape objects created by your application from their original format to a QuickDraw GX stream format, the **shape** is said to be flattened. When the `GXUnFlattenShape` function interprets the data stream of a flattened shape, the shape is said to be unflattened**.**

If QuickDraw GX is available and you need to **flatten** and **unflatten** QuickDraw GX shapes, you just use the `GXFlattenShape` and `GXUnflattenShape` functions. If QuickDraw GX is not available and you need to unflatten a flattened shape, then you need to create an interpreter for the QuickDraw GX data stream that was created when the shape was flattened. The interpreter must be compatible with your current working environment.

Your interpreter needs to parse the data of the QuickDraw GX data stream to extract the original meaning. The format of the data stream is public. This section describes the data **stream format** and its use in print files and portable digital documents.

In addition to the `GXFlattenShape` and `GXUnflattenShape` functions that create and interpret the QuickDraw GX stream format for shapes, there are other flatten and unflatten functions that perform flattening and unflattening operations on job objects, job objects in a handle, collection objects, and fonts. These functions are not directly related to the stream format.

The `GXFlattenJob` and `GXUnFlattenJob` functions provide your application with a mechanism for flattening and unflattening all information associated with a job object by specifying a pointer to a flattening function. For more information on these functions, see the chapters “QuickDraw GX Printing” and “Core Printing Features” in *Inside Macintosh: QuickDraw GX Printing*.

The `GXFlattenJobToHdl` and `GXUnflattenCollectionFromHdl` functions provides your application with a means of flattening and unflattening all information associated with a job object in a handle. For more information on these functions, see the chapters “Introduction to Printing with QuickDraw GX” and “Core Printing Features” of *Inside Macintosh: QuickDraw GX Printing*.

The `GXFlattenCollection` and `GXUnflattenCollection` functions flatten and unflatten information in a collection object. For more information on this function, see the chapter “Collection Manager” in this book.

The `GXFlattenFont` function flattens a font so that it can be included in a flattened shape. The `GXFlattenFont` function is described in the chapter “Font Objects” in *Inside Macintosh: QuickDraw GX Typography.*

Characteristics

The QuickDraw GX data stream format is used whenever a QuickDraw GX shape is stored to disk or printed. Likewise, the data stream must be interpreted whenever the flattened shape is to be used. The QuickDraw GX stream format is

- **Extensible.** The data stream includes type constants called opcodes that specify the meaning of the data that follows in the data stream and record size values that indicate the number of bytes in the record that follow. The opcode and size are always in the same format. If a reader of a QuickDraw GX data stream doesn’t understand the information contained in the stream, the reader can choose to skip to the next opcode. Some opcode constants are reserved for future expansion.
- **Byte oriented.** QuickDraw GX uses a byte-oriented stream format so that it is simple for different processors to interpret the flattened shape information. Multiple byte-oriented data streams, using words (2 bytes) or long words (4 bytes), are larger and therefore are not as efficient for storing, retrieving, and printing shapes.

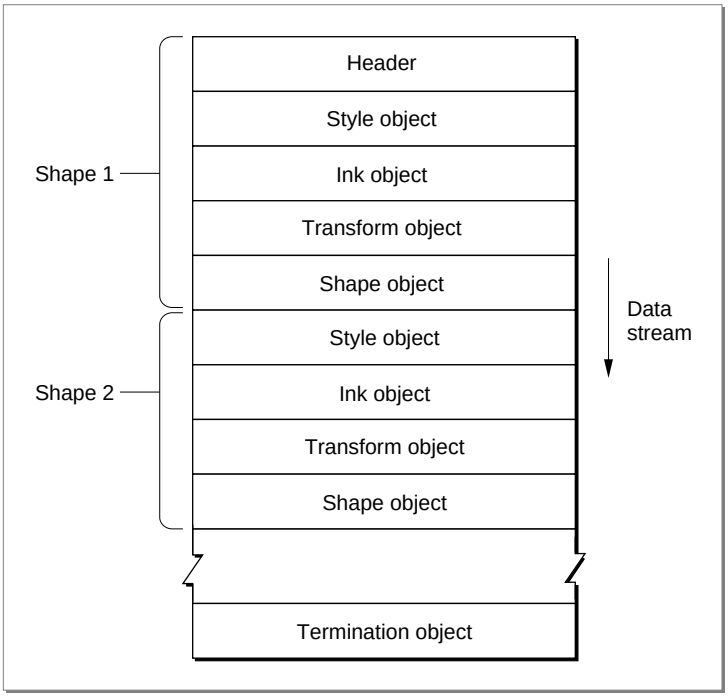
- **Efficient.** The QuickDraw GX data stream format contains a highly structured optimized set of data that minimize the amount of irrelevant information. For example, if your application creates a shape and then moves the shape to another position, the flattened shape stream format describes only the final position of the shape and does not include a description of the intermediate move.
- **Compressed.** The `GXFlattenShape` function always applies a compression algorithm to the flattened shape. The degree of compression that is achieved depends upon the shape and the objects that make up the shape. If applying the compression algorithm results in a data stream that is larger than the original, the original data is adopted as the default. When you call the `GXFlattenShape` function, you are thereby always assured of a data stream format that is equal to or smaller than the original data format. Data in a QuickDraw GX stream format consists of single bits, multiple bits, a byte, multiple bytes, a word, multiple words, a long word, or multiple long words. The QuickDraw GX compression algorithm attempts to minimize the number of bits that are required to represent the data required to describe each object and its properties. For example, the long fixed-point number 125.0, 0x007D0000, requiring 4 bytes may be compressed to the byte 125, 0x7D, requiring only 1 byte. This substitution makes the data stream 3 bytes smaller, while maintaining the integrity of the data value. When the shape is unflattened, the byte must be converted back to its original long value. The QuickDraw GX stream format also compresses the data stream bytes that contain opcodes. These opcode bytes consist of a 2-bit field and a 6-bit field that are packed into1 byte.
- **Shape oriented.** Each QuickDraw GX shape is described by a style object, ink object, transform object, and shape object. When a QuickDraw GX shape is flattened, a new data format is created that contains all of the essential information required to define the original shape. All of the objects and properties that are required to describe all of the QuickDraw GX shapes are included in the data stream.

Stream Design

The data stream includes type constants called *opcodes* that specify the meaning of the data that follows in the data stream and record size values that indicate the number of bytes in the record that follow.

Each QuickDraw GX data stream starts with a header. The header contains the version of QuickDraw GX that produced the stream and flags that describe whether or not a list of fonts and a list of glyphs used by the objects are provided for at the end of the stream. This header is typically followed by the style object, ink object, transform object, and shape object for the shape. This sequence is repeated for all subsequent shapes in the data stream. The data stream is terminated after the last shape by the presence of a termination object, as shown in Figure 7-1.

Figure 7-1 A typical flattened shape data stream sequence



Each header and object type in the data stream is counted. This results in the assignment of **reference** numbers for headers and all object types, such as style, ink, and transform objects. The reference number is the *n*th occurrence of a header or object type.

For example, each data stream always has a header (1), a typically a style object (1), ink object (1), transform object (1), and shape object (1), where the references are given in parentheses. Additional headers and object types in the data stream are assigned the next incremental reference number. Figure 7-1 shows that shape 1 is defined by style object (1), ink object (1), transform object (1), and shape object (1) and that shape 2 is defined by style object (2), ink object (2), transform object (2), and shape object (2). shape 100 in this data stream (not shown) may use the ink object defined in shape 1 by referencing ink object (1).

Besides the style, ink, transform, and shape objects, the data stream may also contain additional objects. The following objects are flattened when referenced by shapes, inks, and transforms:

- tag
- color set
- color profile
- other referenced objects

Examples of other referenced objects are the shapes that represent clips, dashes, and the styles and transforms in text faces.

The following objects are never flattened:

- view ports
- view devices
- view groups

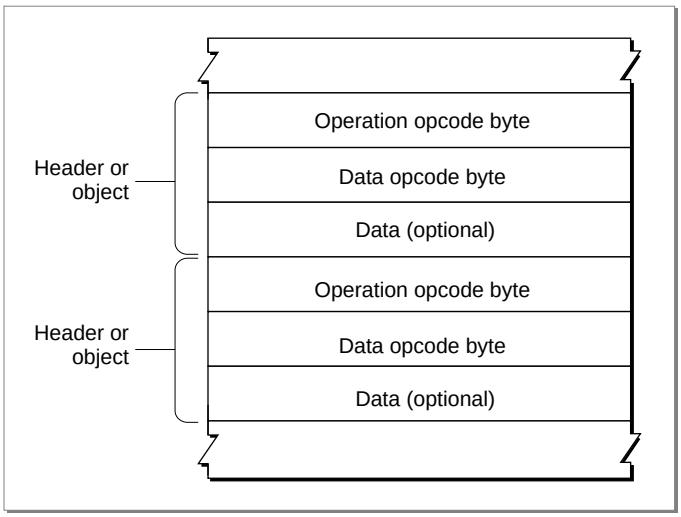
Another rule regarding data stream design requires that all objects and their attributes in the data stream must be defined before they are referenced. QuickDraw GX data streams never forward-reference objects.

For example, the style, ink, and transform objects for a shape must always precede the shape object that they describe in the data stream. In addition, if a style object has a text face property and the text face property has a dash property, then the shape object for the dash property must precede the style object in the data stream.

The data stream design does not require that the order of objects to be style, ink, and transform. Because these objects do not reference each other, they can appear in any order in the data stream, as long as they are defined prior to being referenced.

Each header and object in the data stream consists of an operation opcode byte, a data type opcode byte, and optional data bytes. Figure 7-2 shows these basic data stream format building blocks. This sequence is repeated from the beginning of the stream to the end of the stream. The next sections describe each of these building blocks.

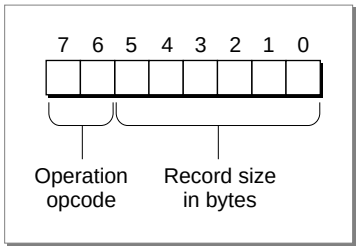
Figure 7-2 Basic components of a stream header or object



Operation Opcode Byte

The first byte of a header or object is always an operation opcode byte. The operation opcode byte contains both an operation opcode and the size in bytes of the record that follows for the current object. The operation opcode either defines a new object, adds data to the current default object, or references a previous object. The record length in bytes includes the data type opcode byte and any data that may follow for the current object. Figure 7-3 shows the format of the operation opcode byte.

Figure 7-3 The format of the operation opcode byte



The operation opcode and record size are always in the same stream format. This enables a reader of the data stream to skip over parts of the data stream that are not understood.

Operation Opcode

Bits 6 and 7 of the operation opcode byte are the operation opcode. Table 7-1 summarizes the 2-bit operation opcodes from the `gxGraphicsOperationOpcode` enumeration.

Table 7-1 Operation opcodes

Type	Value	Description
`gxNewObjectOpcode`	0x00	This opcode type defines a new object.
`gxSetDataOpcode`	0x40	This opcode type adds data to the current object.
`gxSetDefaultOpcode`	0x80	This opcode type replaces the current default with a previously defined object by specifying its reference number.
`gxReservedOpcode`	0xC0	This opcode type is not currently defined and is reserved for future use.

Record Size

The record size defines the number of bytes required to define the header or object record, not including the operation opcode byte. It is always 1 or larger. The record size is given in either bits 0 through 5 of the operation opcode byte or within the bytes that follow the operations opcode byte.

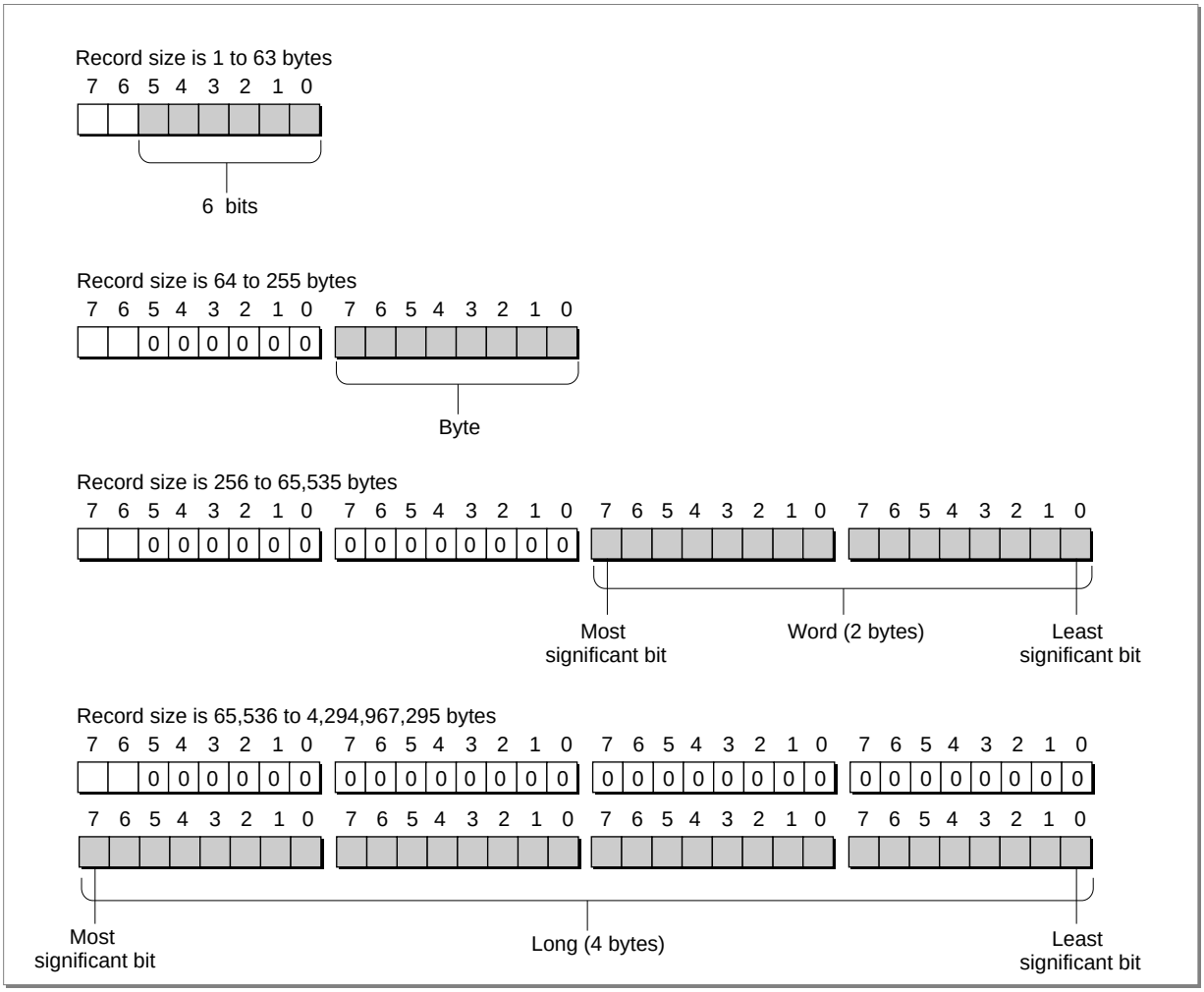
If the record size is larger than the value that can be represented in bits 0 through 5, larger than 63, then a 0 appears in these 6 bits and the next byte in the data stream may contain the record size.

If the record size is larger than the value that can be represented in the next byte, larger than 255, then a 0 appears in this byte and the next word in the stream may contain the record size.

If the record size is larger than the value that can be represented in the next word, larger than 65,535, then a 0 appears in this word and the next long in the stream contains the record size. A long can accommodate a record size up to 4,294,967,295 bytes.

Figure 7-4 shows the operation opcode byte on the left and the subsequent bytes in which the record size is stored in 6-bits, a byte, a `word`, or a `long`. The data stream continues proceeds from left to right.

Figure 7-4 Data format of the record size

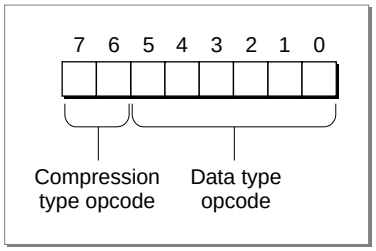


An example of a bit stream in which a long was required to accommodate a record size of 404 bytes is described in the section “Analyzing a Flattened Bitmap Shape” beginning on page 7-81.

Data Type Opcode Byte

A data type opcode byte always follows the record size. This byte contains both a compression type opcode and a data type opcode. Figure 7-5 shows the format of the data type opcode byte.

Figure 7-5 The format of the data type opcode byte



Compression Type Opcode

Bits 6 and 7 of the data type opcode byte contain the compression type opcode. This opcode specifies the type of compression used for the data that follows. The 2-bit compression opcode constants from the `gxTwoBitCompressionValues` enumeration specifies whether the next data are longs, words, bytes, or that no data follows. Table 7-2 lists the compression type opcode values.

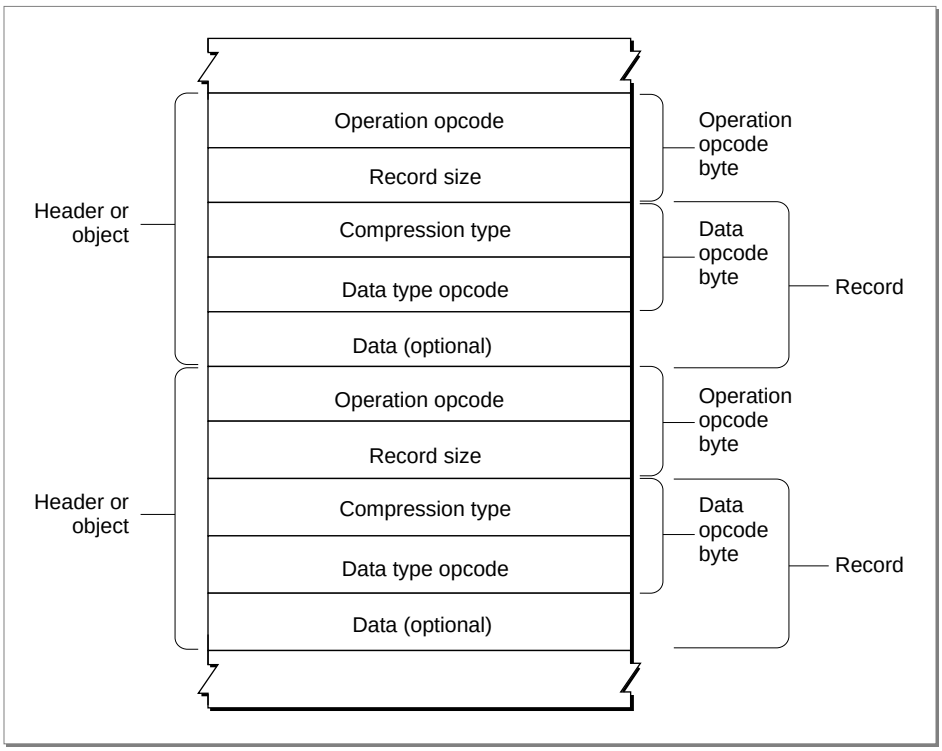
Table 7-2 Compression values

Value	Description
0x00	No compression has been applied. The data that follows are long words.
0x40	Word compression has been applied. The data that follows are words.
0x80	Byte compression has been applied. The data that follows are bytes.
0xC0	Omit compression. No data follows.

The `gxTwoBitCompressionValues` enumeration is also used to interpret the compression in the omit byte. For additional information about the interpretation of omit bytes, see the section “Omit Byte Masks and Omit Byte Shifts” beginning on page 7-22.

The relationship of the operation opcode, record size, compression type opcode, data type opcode, and optional data for a header or object is shown in Figure 7-6.

Figure 7-6 Relationship of stream format components



The appearance or absence of data after the data type opcode byte depends upon the values that appear in the operation opcode byte and the data type opcode byte.

If the `gxNewObjectOpcode` constant appears in the operation opcode byte, a new object follows. The new object copies the default values into the newly created object. The default values may have been changed by the last object created of this type. If the last object and the current object are equal, then the new object requires no additional data for its definition. In this case, the stream following the new opcode byte contains only the compression and data type opcode byte with compression set to no compression.

If the `gxSetDataOpcode` constant appears in the operation opcode byte, the record length is greater than 1 byte and object-specific data follows.

The `gxSetDefaultOpcode` constant appears only after the current object type has been defined. If the `gxSetDefaultOpcode` constant appears in the operation opcode byte, the data type opcode contains the `gxStyleTypeOpcode`, `gxInkTypeOpcode`, or `gxTransformTypeOpcode` constant. The compression type opcode defines the compression of the data of the object reference number that follows. This previously

defined object becomes the default styles, ink, or transform for the shapes created subsequently.

The sequence of the object-specific data that follows the data type opcode byte is described in the next section. Subsections are provided for the header, shape data, style, ink, transform, color profile, color set, tag, bit image, font name, and trailer objects.

Data Type Opcode

Bits 0 through 5 of the data type opcode byte contain the data type opcode. This opcode specifies the type of data that follows. The type of data that follows depends upon the current value of the operation opcode. If the operation opcode is `gxNewObjectOpcode`, the data type opcode describes a new object. These data type opcodes are described in the next section. If the operation opcode is `gxSetDataOpcode`, the data type opcode, specifies how the current object will be modified. These data type opcodes are described in the sections "Data Type Opcodes to Modify a Shape Object" beginning on page 7-17, "Data Type Opcodes to Modify a Color Set Object" beginning on page 7-20, "Data Type Opcodes to Modify a Color Profile Object" beginning on page 7-21, and "Data Type Opcodes to Modify a Transform Object" beginning on page 7-21.

Data Type Opcodes for a New Object

When the current operation opcode is the `gxNewObjectOpcode` constant, bits 0 through 5 of the data type opcode byte specify the data type opcode for the new object. Data type opcode constants for header, style, ink, transform, color profile, color set, tag type, bit image, font name, and trailer are defined in the `gxGraphicsNewOpcode` enumeration. Data type opcode constants for empty, point, line, curve, rectangle, polygon, path, bitmap, text, glyph, layout, full, and picture are defined in the `gxShapeTypes` enumeration. Table 7-3 summarizes all of the data type opcodes for a new object.

Table 7-3 Data type opcodes for a new object

Constant	Value	Description
`gxHeaderTypeOpcode`	0x00	The data that follows is the header.
`gxEmptyType`	0x01	The data that follows describes an empty shape object. See the `GXNewShape(gxEmptyType)` function.
`gxPointType`	0x02	The data that follows describes a point object. See the `GXNewPoint` function.
`gxLineType`	0x03	The data that follows describes a line object. See the `GXNewLine` function.
`gxCurveType`	0x04	The data that follows describes a curve object. See the `GXNewCurve` function.

continued

Table 7-3 Data type opcodes for a new object (continued)

Constant	Value	Description
gxRectangleType	0x05	The data that follows describes a rectangle object. See the GXNewRectangle function.
gxPolygonType	0x06	The data that follows describes a polygon object. See the GXNewPolygons function.
gxPathType	0x07	The data that follows describes a path object. See the GXNewPaths function.
gxBitmapType	0x08	The data that follows describes a bitmap object. See the GXNewBitmap function.
gxTextType	0x09	The data that follows describes a text object. See the GXNewText function.
gxGlyphType	0x10	The data that follows describes a glyph object. See the GXNewGlyph function.
gxLayoutType	0x11	The data that follows describes a layout object. See the GXNewLayout function.
gxFullType	0x12	The data that follows describes a full shape object. See the GXNewShape(gxFullType) function.
gxPictureType	0x13	The data that follows describes a picture object. See the GXNewPicture function.
gxStyleTypeOpcode	0x28	The data that follows describes a style object. See the GXNewStyle function.
gxInkTypeOpcode	0x29	The data that follows describes an ink object. See the GXNewInk function.
gxTransformTypeOpcode	0x2A	The data that follows describes a transform object. See the GXNewTransform function.
gxColorProfileOpcode	0x2B	The data that follows describes a color profile object. See the GXNewColorProfile function.
gxColorSetOpcode	0x2C	The data that follows describes a color set object. See the GXNewColorSet function.
gxTagTypeOpcode	0x2D	The data that follows describes a tag object. See the GXNewTag function.
gxBitImageOpcode	0x2E	The data that follows describes a bit image, the bits pointed to by a bitmap.
gxFontNameTypeOpcode	0x2F	The data that follows describes a font name. See the GXNewFont function.
gxTrailerTypeOpcode	0x3F	This opcode indicates the end of a data stream.

The omitted numbers are reserved by Apple Computer, Inc. for future use. You should extend the stream format by using tag objects to encapsulate custom data. Tags are described in the "Tag Objects" in *Inside Macintosh:QuickDraw GX Objects.*

Data Type Opcodes to Modify a Shape Object

When the current object is a shape object and the current operation opcode is the `gxSetDataOpcode` constant, bits 0 through 5 of the data type opcode byte specify the data type opcode for the shape object to be modified. Data type opcode constants for attributes, tag, ink, and fill are defined in the `gxShapeDataOpcode` enumeration. Table 7-4 summarizes all of the data type opcodes used to modify a shape object.

Table 7-4 Data type opcodes to modify a shape object

Constant	Value	Description
`gxShapeAttributesOpcode`	0x00	The attributes data that follows is added to the current shape object. See the `GXSetShapeAttributes` function.
`gxTagOpcode`	0x01	The tag data that follows is added to the current shape object. See the `GXSetShapeTags` function.
`gxFillOpcode`	0x02	The fill data that follows is added to the current shape object. See the `GXSetShapeFill` function.

Data Type Opcodes to Modify a Style Object

When the current object is a style object and the current operation opcode is the `gxSetDataOpcode` constant, bits 0 through 5 of the data type opcode byte specify the data type opcode for the style object to be modified. Data type opcode constants for attributes, tag, curve error, pen, join, dash, caps, pattern, text attributes, text size, font, text face, platform, font variations, run controls, run priority justification override, run glyph justification overrides, run glyph substitutions, run features, run kerning adjustments, and justification are defined in the `gxStyleDataOpcode` enumeration. Table 7-5 summarizes all of the data type opcodes used to modify a style object.

Table 7-5 Data type opcodes to modify a style object

Constant	Value	Description
gxStyleAttributesOpcode	0x00	The attributes data that follows is added to the current shape object. See the GXSetStyleAttributes function.
gxStyleTagOpcode	0x01	The tag data that follows is added to the current shape object. See the GXSetStyleTags function.
gxStyleCurveErrorOpcode	0x02	The curve error data that follows is added to the current style object. See the GXSetStyleCurveError function.
gxStylePenOpcode	0x03	The pen data that follows is added to the current style object. See the GXSetStylePen function.
gxStyleJoinOpcode	0x04	The join data that follows is added to the current style object. See the GXSetStyleJoin function.
gxStyleDashOpcode	0x05	The dash data that follows is added to the current style object. See the GXSetStyleDash function.
gxStyleCapsOpcode	0x06	The caps data that follows is added to the current style object. See the GXSetStyleCaps function.
gxStylePatternOpcode	0x07	The pattern data that follows is added to the current style object. See the GXSetStylePattern function.
gxStyleTextAttributesOpcode	0x08	The text attributes data that follows is added to the current style object. See the GXSetStyleTextAttributes function.
gxStyleTextSizeOpcode	0x09	The text size data that follows is added to the current style object. See the GXSetStyleTextSize function.
gxStyleFontOpcode	0x0A	The font data that follows is added to the current style object. See the GXSetStyleFont function.

Table 7-5 Data type opcodes to modify a style object (continued)

Constant	Value	Description
gxStyleTextFaceOpcode	0x0B	The text face data that follows is added to the current style object. See the GXSetStyleFace function.
gxStylePlatformOpcode	0x0C	The platform data that follows is added to the current style object. See the GXSetStyleEncoding function.
gxStyleFontVariationsOpcode	0x0D	The font variations data that follows is added to the current style object. See the GXSetStyleFontVariations function.
gxStyleRunControlsOpcode	0x0E	The run controls data that follows is added to the current style object. See the GXSetStyleRunControls function.
gxStyleRunPriorityJustOverrideOpcode	0x1F	The run priority justification override data that follows is added to the current style object. See the GXSetStyleRunPriorityJust Override function.
gxStyleRunGlyphJustOverridesOpcode	0x10	The run glyph justification overrides data that follows is added to the current style object. See the GXStyleRunGlyphJust Overrides function.
gxStyleRunGlyphSubstitutionsOpcode	0x11	The run glyph substitutions data that follows is added to the current style object. See the GXStyleRunGlyphSubstitutions function.
gxStyleRunFeaturesOpcode	0x12	The run features data that follows is added to the current style object. See the GXStyleRunFeatures function.
gxStyleRunKerningAdjustmentsOpcode	0x13	The run kerning adjustments data that follows is added to the current style object. See the GXStyleRunKerning Adjustments function.
gxStyleJustificationOpcode	0x14	The justification data that follows is added to the current style object. See the GXStyleJustification function.

Data Type Opcodes to Modify an Ink Object

When the current object is an **ink** object and the current operation opcode is the `gxSetDataOpcode` constant, bits 0 through 5 of the data type opcode byte specify the data type opcode for the ink object to be modified. Data type opcode constants for attributes, tag, color, and transfer mode are defined in the `gxInkDataOpcode` enumeration. Table 7-6 summarizes all of the data type opcodes used to modify an ink object.

Table 7-6 Data type opcodes to modify an ink object

Constant	Value	Description
`gxInkAttributesOpcode`	0x00	The attributes data that follows is added to the current ink object. See the `GXSetInkAttributes` function.
`gxInkTagOpcode`	0x01	The tag data that follows is added to the current ink object. See the `GXSetInkTags` function.
`gxInkColorOpcode`	0x02	The ink color data that follows is added to the current ink object. See the `GXSetInkColor` function.
`gxInkTransferModeOpcode`	0x03	The ink transfer mode data that follows is added to the current ink object. See the `GXSetInkTransfer` function.

Data Type Opcodes to Modify a Color Set Object

When the current object is a color set object and the current operation opcode is the `gxSetDataOpcode` constant, bits 0 through 5 of the data type opcode byte specify the data type opcode for the color set object to be modified. A data type opcode constant for tag is defined in the `gxColorSetDataOpcode` enumeration. The constant 0 is reserved for future use. Table 7-7 summarizes all of the data type opcodes used to modify a color set object.

Table 7-7 Data type opcodes to modify a color set object

Constant	Value	Description
`gxColorSetReservedOpcode`	0x00	This constant is reserved for future assignment.
`gxColorSetTagOpcode`	0x01	The tag data that follows is added to the current color set object. See the `GXSetColorSetTags` function.

Data Type Opcodes to Modify a Color Profile Object

When the current object is a color profile object and the current operation opcode is the `gxSetDataOpcode` constant, bits 0 through 5 of the data type opcode byte specify the data type opcode for the color profile object to be modified. A data type opcode constant for tag is defined in the `gxProfileDataOpcode` enumeration. The constant 0 is reserved for future use. Table 7-8 summarizes the data type opcodes used to modify a color profile object.

Table 7-8 Data type opcodes to modify a color profile object

Constant	Value	Description
`gxColorProfileReservedOpcode`	0x00	This constant is reserved for future assignment.
`gxColorProfileTagOpcode`	0x01	The tag data that follows is added to the current color profile object. See the `GXSetColorProfileTags` function.

Data Type Opcodes to Modify a Transform Object

When the current object is a **transform** object and the current operation opcode is the `gxSetDataOpcode` constant, bits 0 through 5 of the data type opcode byte specify the data type opcode for the transform object to be modified. A data type opcode constant for tag is defined in the `gxTransformDataOpcode` enumeration. The constant 0 is reserved for future use. Table 7-9 summarizes the data type opcodes used to modify a transform object.

Table 7-9 Data type opcodes to modify a transform object

Constant	Value	Description
`gxTransformReservedOpcode`	0x00	This constant is reserved for future assignment.
`gxTransformTagOpcode`	0x01	The tag data that follows is added to the current transform object. See the `GXSetTransformTags` function.
`gxTransformClipOpcode`	0x02	The tag data that follows is added to the current transform object. See the `GXSetTransformClip` function.
`gxTransformMappingOpcode`	0x03	The tag data that follows is added to the current transform object. See the `GXSetTransformMapping` function.

continued

Table 7-9 Data type opcodes to modify a transform object (continued)

Constant	Value	Description
`gxTransformPartMaskOpcode`	0x04	The tag data that follows is added to the current transform object. See the description of the `gxShapePart` mask parameter to the `GXSetTransformHitTest` function.
`gxTransformToleranceOpcode`	0x05	The tag data that follows is added to the current transform object. See the description of the Fixed tolerance parameter to the `GXSetTransformHitTest` function.

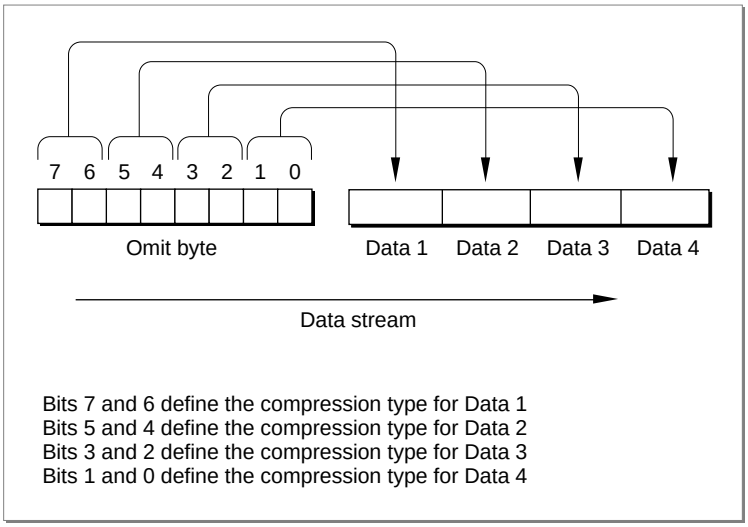
Data

The sequence of the optional object-specific data that follows a data type opcode byte is predetermined and consists of type constants and data. Some data sequences are preceded by an omit byte. An **omit byte** is included in the data stream format to describe the presence or absence, meaning, order, and compression of data that corresponds to the fields of a type or the properties of an object. If an omit byte is not present for an object, then, with the exception of bitmaps and transforms, the compression type opcode in the data type opcode byte defines the data compression.

Omit Byte Masks and Omit Byte Shifts

The omit byte provides an efficient method of assigning different data compressions to type constants and object properties that immediately follow the omit byte. Figure 7-7 shows the relationship of the bits in an omit byte and the four constants or properties that follow.

Figure 7-7 Omit byte relationship with the data that follows



The compression type constants used in the omit byte are defined in the `gxTwoBitCompressionValues` enumeration listed in Table 7-2. Long, word, or byte data compression is applied if the enumeration constants are 0x00, 0x40, 0x80, respectively. If the constant is 0xC0, the compression is “omit compression,” then the stream format does not include the field or property. For example, if the omit byte in Figure 7-7 contained 0x0C for bits 3 and 2, Data 3 constant or property would not appear in the stream and Data 4 would follow Data 2.

Some omit byte enumerations provide multiple bytes of mask constants and shift constants to accommodate the description of all of the properties of an object or all of the fields of a structure. For example, the description of a layout shape requires three omit bytes to specify the compression of all of the properties. The data corresponding to each omit byte mask follows the mask. For multiple masks, the sequence is omit mask1, data, omit mask2, data, omit mask3, data, and so on.

You can use an **omit byte mask** and its corresponding **omit byte shift** to interpret the meaning of each of the bits in the omit byte. Each entry in an omit mask enumeration has a name and a value. The name describes the property. The hexadecimal value of the mask is given in the enumeration. The binary equivalent is the mask.

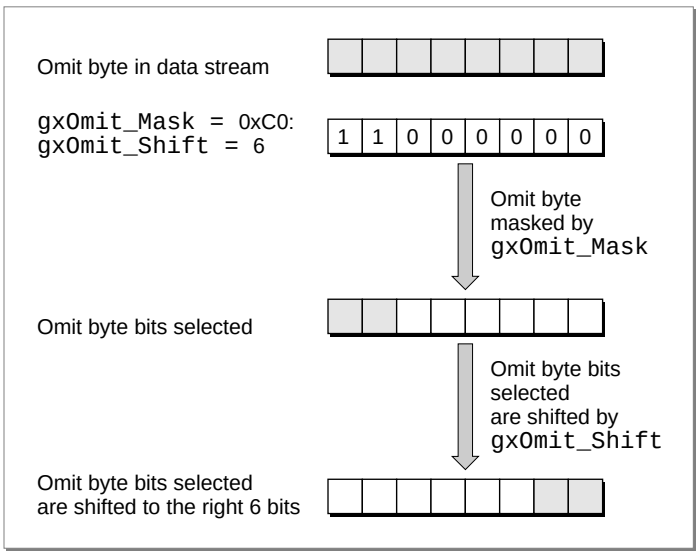
Table 7-10 shows a typical omit byte mask enumeration and its corresponding omit byte shift enumeration values. The example shows the `gxOmitTextMask` enumeration binary mask values and the bit shift from the corresponding `gxOmitTextShift` enumeration.

Table 7-10 Constants from the `gxOmitTextMask` and the `gxOmitTextShift` enumerations

gxOmitTextMask enumeration	Enumeration value	Binary mask value	Bit shift constant
`gxOmitTextCharacterMask`	`0xC0`	11000000	6
`gxOmitTextPositionXMask`	`0x30`	00110000	4
`gxOmitTextPositionYMask`	`0x0C`	00001100	2
`gxOmitTextDataMask`	`0x02`	00000010	1

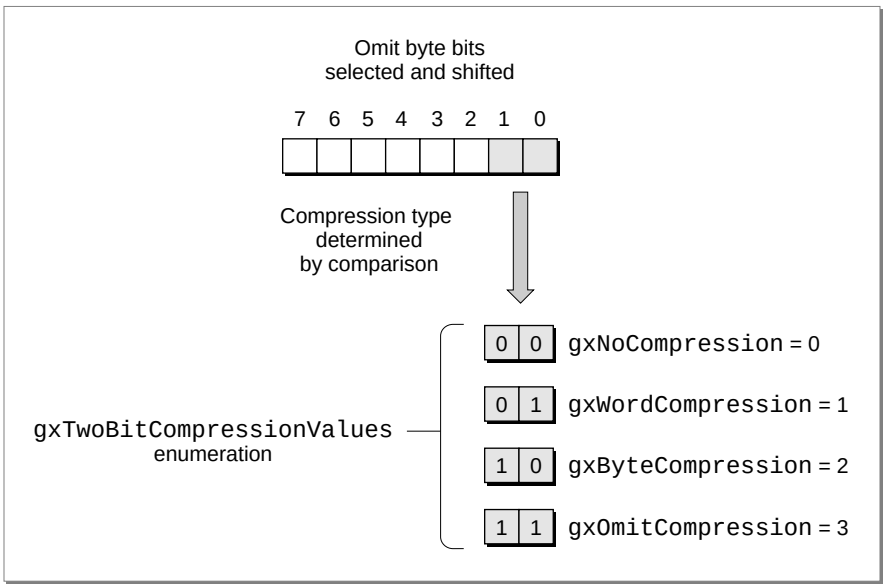
Figure 7-8 shows how you can use an omit mask and corresponding omit shift to analyze an omit byte in the data stream.

Figure 7-8 Select the bits from the omit byte



First, the bits in the omit byte are masked with the `gxOmit_Mask` enumeration with a value of 0xC0 and a binary value 11000000. This mask selects the first two high-order bits of the omit byte. In order to interpret the two bits selected, shift the bits to the right by the number of bits indicated by the `gxOmit_Shift` enumeration value. Once the bits are selected and shifted, determine the compression of the data that follows by comparing these bits with the `gxTwoBitCompressionValues` enumeration, as shown in Figure 7-9. The values of the `gxTwoBitCompressionValues` enumeration are given in Table 7-2.

Figure 7-9 Compare the bits selected and shifted with the compression enumeration



Here is an example of how this works with an omit byte describing the shape object for a text shape. First you need to correlate the names of the constants in the omit mask enumeration with the structures, enumerations, or properties of the object that they describe. For more information on correlating omit bytes, see the appropriate object-specific heading in the section“Data” beginning on page 7-22.

Table 7-11 shows the correlation between the gxOmitTextMask names and the parameters of the GXNewTextFunction.

Table 7-11 Correlation between gxOmitTextMask and the GXNewText function

Constants in the gxOmitTextMask enumeration	Text shape property
gxOmitTextCharacterMask	charCount
gxOmitTextPositionXMask	position .x
gxOmitTextPositionYMask	position .y
gxOmitTextDataMask	text

A summary of these constants is provided in Table 7-10. The gxOmitTextMask enumeration constants correlate with the properties of the text shape. The text shape is described in the text shape chapter of *Inside Macintosh: QuickDraw GX Typography.*

The order of the gxOmitTextMask enumeration tells us that the data to follow will be in the sequence charCount, position .x, position .y, and text.

For instance, suppose the omit byte is 0xA4 or binary 10100100.

The binary mask value for the gxOmitTextCharacterMask, 11000000, selects the high order 2 bits, 10. The gxTwoBitCompressionValues enumeration with value 2 is gxByteCompression. The data for charCount is therefore byte compressed.

The binary mask value for the gxOmitPositionXMask, 00110000, selects the next 2 bits, 10. The gxTwoBitCompressionValues enumeration with value 2 is again gxByteCompression. The data for position.x is therefore byte compressed.

The binary mask value for the gxOmitPositionYMask, 00001100, selects the next 2 bits, 01. The gxTwoBitCompressionValues enumeration with value 1 is gxWordCompression. The data for position.y is therefore word compressed.

The binary mask value for the gxOmitTextDataMask, 10, selects the next bit, 0. The gxTwoBitCompressionValues enumeration with value 0 is gxNoCompression. The text data is therefore not compressed.

The above example is from the analysis of a data stream of a flattened text shape. For additional information about this example see the section “Analyzing a Flattened Text Shape” beginning on page 7-72.

One or more omit mask bytes are included in the data stream whenever specific enumeration or structure data is required to describe a specific object.

Omit mask and omit shift enumerations can be used to analyze QuickDraw GX omit bytes and compare the masked bits to other values.

An omit byte is first masked to obtain the bits desired. The bits are then shifted using the omit shift enumeration that corresponds to the omit byte. The resulting bits can then be compared to other data in your application to obtain information about the data stream.

Listing 7-1 shows how to determine if the x-coordinate of the position field in a flattened shape data stream is compressed.

Listing 7-1 Determining if `position(x)` is byte compressed

```
unsigned char a = ReadByte();
if ((a & (gxOmitTextPositionXMask >> gxOmitTextPositionXShift)) ==
    gxByteCompression
{

/* perform an action */

}
```

The function reads the byte, masks it with `gxOmitTextPositionXMask` to obtain the desired two bits, and then shifts it by the amount given by the `gxOmitTextPositionShift`. The resulting 2 bits can now be compared to the 2 bits of `gxByteCompression`.

Header Data

The header marks the beginning of a new flattened shape in the data stream. The `gxHeaderTypeOpcode` constant indicates that the version of QuickDraw GX that generated the data stream follows. As new versions become available, older software may not be able to interpret the newer portions of a data stream. The interpreter can then look at the version number and skip over versions that it doesn't understand. For example, if an interpreter that understands only QuickDraw GX version 1.0 encounters version 2.0 or if the interpreter finds a version 1.0 opcode, but doesn't recognize the data, an error is posted.

The byte after the version byte contains the `gxFontListFlatten` and `gxFontGlyphsFlatten` flags. These flags are functional only if the shape contains text.

The `gxFontListFlatten` flag instructs the `GXFlattenShape` function to attach a tag object to the flattened shape containing a list of the fonts referenced in the shape. A list of all of the fonts used in the data stream are included at the end of the data stream.

The `gxFontGlyphsFlatten` flag instructs the `GXFlattenShape` function to attach a tag to the flattened shape containing a list of the specific glyphs used from each font referenced by the shape. A list of all of the glyph codes used by all of the fonts referenced in a data stream is then included at the end of that data stream.

For more information about the font and glyph list flags, see the chapter "Shape Objects" in *Inside Macintosh: QuickDraw GX Objects.*

The font list and glyph list are combined to form a tag that is of type `gxFlatFontListItem` and designated `'flst'`. During printing, only the fonts and glyphs used in the stream are loaded to the printing device.

The `gxFlatFontList` structure includes the `gxFlatFontListItem` structure. The `gxFlatFontListItem` contains two arrays. The first is the array of font names. The second is the array of glyphs that are used. The array of glyphs is obtained by setting a bit in an array for each glyph that is used. If you ask only for the font names, the glyph array will be omitted. The glyphs array cannot be selected without the font array selected. In other words, you may specify either a list of fonts or specify a list of fonts and glyphs to be listed at the end of the data stream.

The fonts and glyphs included in the flattened list, `'flst'`, are used in the print file for the QuickDraw GX portable digital document. For more information on the QuickDraw GX portable digital document see the section "Portable Digital Documents" beginning on page 7-53.

For more information on the QuickDraw GX print file, see the section "About Print Files and Portable Digital Documents" beginning on page 7-51. For more information about how to use the print file data, see the section "Obtaining Data From a Print File" beginning on page 7-89.

For more information on the `gxFlatFontName`, `gxFlatFontListItemTag`, and `gxFlatFontList` structures see the chapter "Fonts" in *Inside Macintosh: QuickDraw GX Typography.*

New Shape Object Data

A new shape object always follows the style, ink, transform, and any other objects that have been built for the shape object in the data stream. New shape data follows an operation opcode `gxNewObjectOpcode` constant and a data type opcode containing one of the constants in the `gxGraphicsNewOpcode` enumeration. Values 1 (`gxEmptyType`) through 13 (`gxPictureType`) are the constants from the `gxShapeTypes` enumeration.

This opcode creates a new shape object with all of the properties of the previous shape object in the data stream. If the current shape object is the first shape object in the stream, then it is created with default properties.

The values of the constants for all of the shape objects are summarized in Table 7-3. Shape types are described in the chapter "Shape Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Empty Shape Data

The data type opcode with a value 1 is the `gxEmptyType` constant. Empty shapes store no information in their geometries. For the current shape object, the `gxEmptyType` means that the current shape is an empty shape. No data follows.

The `gxEmptyTypes` constant is described in the chapter "Geometric Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Point Shape Data

The data type opcode with a value 2 is the `gxPointType` constant. The data for the fields of a `gxPoint` structure follows. The data sequence is `x` (`Fixed`), `y` (`Fixed`).

Compression data: `gxNoCompression` - read 2 `longs` per point; `gxWordCompression` - read 2 shorts per point or treat each short as a signed integer (120 = 120.0 and –171 = –171.0); `gxByteCompression` - read 2 bytes per point and treat each byte as a signed integer (7 = 7.0 and –13 = –13.0).

The `gxPoint` structure is described in the chapter "Geometric Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Line Shape Data

The data type opcode with a value 3 is the `gxLineType` constant. The data for the fields of the `gxLine` structure follows. The data sequence is `first.x`, `first.y` `last.x`, `last.y`.

Compression data: `gxNoCompression` - read 2 `longs` per point; `gxWordCompression` - read 2 shorts per point or treat each short as a signed integer (120 = 120.0 and –171 = –171.0); `gxByteCompression` - read 2 bytes per point and treat each byte as a signed integer (7 = 7.0 and –13 = –13.0).

The `gxLine` structure is described in the chapter "Geometric Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Curve Shape Data

The data type opcode with a value 4 is the `gxCurveType` constant. The data for the fields of the `gx`Curve structure follows. The fields in the structure correspond to the parameters in the `GXNew`Curve function. The data sequence is `x` (first point), `y` (first point), `x` (control point), `y` (control point), `x` (last point), and `y` (last point).

Compression data: `gxNoCompression` - read 2 `longs` per point; `gxWordCompression` - read 2 shorts per point or treat each short as a signed integer (120 = 120.0 and –171 = –171.0); `gxByteCompression` - read 2 bytes per point and treat each byte as a signed integer (7 = 7.0 and –13 = –13.0).

The `gxCurve` structure is described in the chapter "Geometric Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Rectangle Shape Data

The data type opcode with a value 5 is the `gxRectangleType` constant. The data for the fields of the `gxRectangle` structure follows. The data sequence is left, top, right, bottom. Typically, the first corner is left-top and the second corner is right-bottom; but this order is not required. They need only be opposite corners of a rectangle.

Compression data: gxNoCompression - read 2 longs per point; gxWordCompression - read 2 shorts per point or treat each short as a signed integer (120 = 120.0 and –171 = –171.0); gxByteCompression - read 2 bytes per point and treat each byte as a signed integer (7 = 7.0 and –13 = –13.0).

The gxRectangle structure is described in the chapter "Geometric Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Polygon Shape Data

The data type opcode with a value 6 is the gxPolygonType constant. The data for the fields of the gxPolygons structure follows. The gxPolygons structure includes the gxPolygon structure.

The data sequence is contours, vectors, omit byte, x (first point), y (first point), x (second point), y (second point), x (third point), y (third point), and so on. The numbers are compressed as fixed-point numbers.

The point array for polygons and paths stream is stored as relative positions, not absolute positions (as is the case for the point arrays in polygon and path shapes.)

The omit byte is interpreted by the gxOmitPathMask and gxOmitPathShift enumerations.

The first two entries of the omit byte describe the compression for the first two points of the polygon shape, which are absolute. The numbers are compressed as fixed-point numbers: gxNoCompression means 1 long for each fixed number; gxWordCompression means 1 short for each fixed number treated as an integer (17 = 17.0); gxByteCompression means 1 byte per fixed number. Thus a byte compressed value can represent an integer fixed point number from –128.0 to 127.0; a word compression value can represent any integer fixed-point number.

The second two entries in the omit byte describe the compression for the second through the last points in the contour. The coordinates of these points are relative to the first absolute points and appear in the stream as differences. The relative values are stored as differences. Thus each x value in the stream is subtracted from the prior value to reconstruct the original value. Conversely, each value in the shape is subtracted from the prior value to compute the delta to be written to the stream. The x and y coordinate values are considered separately. Each may be independently byte, word, or long compressed, using the same fixed-point compression as the absolute values. Each subsequent contour has its own omit byte to describe the absolute initial point values and the subsequent relative point values.

The compression bits in the data type opcode byte control the compression of the contour counts and all vector counts. Compression data: gxNoCompression - read 1 long for contour and each vector count; gxWordCompression - read 1 word for contour count and each vector count; gxByteCompression - read 1 byte for contour count and each vector count.

The gxPolygons structure is described in the chapter "Geometric Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Path Shape Data

The data type opcode with a value 7 is the `gxPathType` constant. The data for the fields of the gxPaths structure follows. The `gxPaths` structure includes the `gxPath` structure.

The data sequence is `contours` (number of contours), `vectors` (number of points in the contour), control bytes, omit byte, `x` (absolute coordinate of first point), `y` (absolute coordinate of first point), `x` (relative coordinate of second point), `y` (relative coordinate of second point), `x` (relative coordinate of third point), `y` (relative coordinate of third point), and so on.

A control byte contains control bits for each point off or on the path. Each point is assigned a bit. Bits with value 1 are off the path; bits with value 0 are on the path. If the number of points exceeds 8, multiple control bytes are used. If the number of points is not an even multiple of 8, the final unused bits are ignored.

The omit byte is interpreted by the `gxOmitPathMask` and `gxOmitPathShift` enumerations.

The first two entries of the omit byte describe the compression for the first two points of the path shape, which are absolute coordinates. The numbers are compressed as fixed-point numbers: `gxNoCompression` means 1 `long` for each fixed number; `gxWordCompression` means 1 `short` for each fixed number treated as an integer (17 = 17.0); `gxByteCompression` means 1 byte per fixed number. Thus a byte compressed value can represent an integer fixed point number from –128.0 to 127.0; a word compression value can represent any integer fixed-point number.

The second two entries in the omit byte describe the compression for the second through the last relative points in the contour. The coordinates of these points are relative to the first absolute points and appear in the stream as differences. Thus each x value in the stream is subtracted from the prior value to reconstruct the original value. Conversely, each value in the shape is subtracted from the prior value to compute the delta to be written to the stream. The x and y coordinate values are considered separately. Each may be independently byte, word, or long compressed, using the same fixed-point compression as the absolute values. Each subsequent contour has its own omit byte to describe the absolute initial point values and the subsequent relative point values.

The compression bits in the data type opcode byte control the compression of the contour counts and all vector counts. Compression data: `gxNoCompression` - read 1 long for contour and each vector count; `gxWordCompression` - read 1 word for contour count and each vector count; `gxByteCompression` - read 1 byte for contour count and each vector count.

The `gxPaths` structure is described in the chapter “Geometric Shapes” in *Inside Macintosh: QuickDraw GX Graphics.*

Bitmap Shape Data

The data type opcode with a value 8 is the `gxBitmapType` constant. The data for the fields of the `gxBitmap` and `gxPoint` structures follow. The `gxBitmap` structure includes the `gxColorSpace` enumeration and the references to the `gxColorSet` and `gxColorProfile` structures.

The data sequence is omit byte 1, `image` reference, `width`, `height`, `rowBytes`, omit byte 2, `pixelSize`, `space` (color space), `set` (color set), `profile` (color profile), omit byte 3, `x` (position), `y` (position).

Omit byte 1 is interpreted by the `gxOmitBitmapMask1` and `gxOmitBitmapShift1` enumerations. Omit byte 2 is interpreted by the `gxOmitBitmapMask2` and `gxOmitBitmapShift2` enumerations. Omit byte 3 is interpreted by the `gxOmitBitmapMask3` and `gxOmitBitmapShift3` enumerations.

Data compression: The value may be a byte, word, or long. The value references a previous bit image: a value of 1 references the first bit image, a value of 2 references the second bit image, etc. A value of 0 indicates that the bitmap references a bit image through a file alias. The bitmap shape must reference a tag containing the file alias and offset as described in the chapter "Tag Objects" in *Inside Macintosh: QuickDraw GX Objects.* All bitmap values are compressed as integers (see polygon coutour compression above) except for the x and y coordinate positions. These are compressed as `Fixed` (see polygon first absolute position). Unlike prior shape types in this section, bitmaps and shape types described below can also have fields with the `gxOmitCompression` bits set. In this case, the value 0 or `nil` is used wherever the omit compression bits are set.

The `gxBitmap` structure is described in the chapter "Bitmap Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Text Shape Data

The data type opcode with a value 9 is the `gxTextType` constant. The data that follows corresponds to the parameters of the `GXNewText` function.

The data sequence is omit byte, byte length (of text), `x` (position), `y` (position), `charCount` (number of characters), data (character text).

The data is the character stream or glyph indexes for the text. For nonRoman scripts, the actual byte length may be more than the number of characters.

The omit byte is interpreted by the `gxOmitTextDataMask` and `gxOmitTextDataShift` enumerations.

Data compression: The byte length is compressed as a long. The x and y coordinates are compressed as a fixed number. The data stream may contain bytes or shorts. If the stream contains shorts and all values are less than 255, then the stream may be compressed. It is an error to specify a character count of zero (omit compression) and to set the text data omit bit.

The `GXNewText` function is described in the chapter "Text Shapes" in *Inside Macintosh: QuickDraw GX Typography.*

Glyph Shape Data

The data type opcode with a value 10 is the `gxGlyphType` constant. The data correspond to the parameters in the `GXNewGlyphs` function and include the `gxPoint` and `gxStyle` structures.

The data sequence is omit byte 1, `charCount` (number of characters), byte length (of text), `runNumber` (number of runs), data (glyph character), omit byte 2, `positions`, `advance`, `tangents`, `styleRuns`, `glyphStyles`.

Omit byte 1 is interpreted by the `gxOmitGlyphDataMask1` and `gxOmitGlyphDataShift1` enumerations. Omit byte 2 is interpreted by the `gxOmitGlyphDataMask2` and `gxOmitGlyphDataShift2` enumerations.

Data compression: `charCount`, byte length, and `runNumber` are compressed as longs. If `charCount` is 0, the data, positions, advance, and tangents are not read. If the `gxOmitGlyphOnePosition` bit is set in the first byte, then the glyph shape contains 1 absolute position or as many positions as there are in the stream. In either case, all are compressed as fixed point values, as bytes, words, or longs. Unlike polygon positions, the x and y values do not have separate compression bits, nor are the positions stored in the relative manner of polygons or paths.

The advances in the glyph shape are read after the positions, if the `gxOmitGlyphAdvance` bits are not `gxOmitCompression` constant. The character count determines the number of bytes read, as is the case with the control bits in a path shape.

If the `gxOmitGlyphTangent` bits in the second omit byte are not equal to the `gxOmitCompression` constant, the `tangents` parameter follows. The tangent values are stored and compressed identically to the positions. If the number of runs (`runNumber`) is greater than zero, then 1 bit in the second omit byte interprets the runs as shorts or shorts compressed to bytes (like the text character compression). If `runNumber` is greater than 0, then the style array is compressed into an array of bytes, words, or longs. The values are references to previous styles in the stream: a value of 1 references the 1st style in the stream, and so on.

The `GXNewGlyphs` function is described in the chapter “Glyph Shapes” in *Inside Macintosh: QuickDraw GX Typography.*

Layout Shape Data

The data type opcode with a value 11 is the `gxGlyphType` constant. The data correspond to the parameters in the `GXNew`Layout function.

Layouts are compressed in a way that is similar to glyphs. Like all types that are greater than or equal to `bitmap` type, all fields default to zero and omit compression is allowed. If the length is greater than 0, the data is read as shorts compressed as bytes or as an uncompressed stream (like text and glyphs). If the style run number is greater than 0, the style run array and style array are present identically to the glyph format. If the `omitLayoutHasBaseline` bit is set in omit byte 3, uncompressed data is read the size

of the gxLineBaselineRecord. If the level run number is greater than zero, the 4th omit byte (read regardless) specifies the compression of the levelRunLength and level arrays as an optionally compressed array of shorts.

The data sequence is omit byte 1, length, x (position), y (position), data, omit byte 2, width, flush, set, just, options, omit byte 3, style, run number, level run number, hasBaseline, style runs, styles, omit byte 4, level runs, levels.

Omit byte 1 is interpreted by the gxOmitLayoutMask1 and gxOmitLayoutShift1 enumerations. Omit byte 2 is interpreted by the gxOmitLayoutMask2 and gxOmitLayoutShift2 enumerations. Omit byte 3 is interpreted by the gxOmitLayoutMask3 and gxOmitLayoutShift3 enumerations. Omit byte 4 is interpreted by the gxOmitLayoutMask4 and gxOmitLayoutShift4 enumerations.

The GXNewLayout function is described in the chapter "Layout Shapes" in *Inside Macintosh: QuickDraw GX Typography.*

Full Shape Data

The data type opcode with a value 12 is the gxFullType constant. Full shapes store no information in their geometries. For the current shape object, the gxFullType constant is a parameter in the GXNewShape function. No data follows.

The gxFullType constant is described in the chapter "Geometric Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Picture Shape Data

The data type opcode with a value 13 is the gxPictureType constant. The data corresponds to the parameters in the GXNewPicture function. The data sequence is omit byte 1, the number of items (compressed as long as specified by the data type opcode), followed by an array of shapes and optional arrays of styles, inks, and transforms. The shape array must exist and may not contain nil (zero) references. The styles, inks and transform array references may be omitted entirely.

The gxPicture structure is described in the chapter "Picture Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

Modified Shape Object Data

Once a shape object is defined, it can be modified. Modified shape data follow a gxSetDataOpcode operation opcode and a data type opcode containing one of the constants from the gxShapeDataOpcode enumeration. Table 7-4 summarizes the values of the constants for all of the modified shape objects.

Attributes Data

An attribute is added to the current shape object if the data type opcode has value 0. This is the `gxShapeAttributesOpcode` constant.

The data for the fields of the `gxShapeAttributes` structure follow and are compressed as long. That data may be 1, 2, or 4 bytes depending on the compression bits.

The `gxShapeAttributes` enumeration is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects.*

Tag Data

A tag is added to the current shape if the data type opcode has value 1. This is the `gxShapeTagOpcode` constant. The data for the parameters of the `GXSetShapeTags` function follows.

The size of the opcode specifies the number of tags; the compression specifies whether the data is in bytes, words, or longs. For instance, if the size is 4 and the compression is `gxShortCompression` (2 bytes), then the stream contains 4/2 == 2 tags. The equivalent operation would be `GXSetShapeTags` (shape, nil, 1, 0, 2, tag array).

The `GXSetShapeTags` function is described in the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects.*

Fill Data

A **shape fill,** compressed as long, is added to the current shape if the data type opcode has value 2. This is the `gxShapeFillOpcode`. A constant from the `gxShapeFill` enumeration follows.

The `gxShapeFills` enumeration is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects.*

New Style Object Data

Data for a new style object follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 28. This is the `gxStyleTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration.

This opcode creates a new style object with all of the properties of the previous style object in the data stream. If the current style object is the first style object in the stream, then it is created with default properties. No data follows for the new style object.

The style object is described in the chapter “Style Objects” in *Inside Macintosh: QuickDraw GX Objects.*

Modified Style Object Data

Once a style object is defined, it can be modified by the addition of style data. Modified style data follows a `gxSetDataOpcode` operation opcode and a data type opcode containing one of the constants from the `gxStyleDataOpcode` enumeration. Table 7-5 summarizes the values of the constants for all of the modified style objects. For all style data, the opcodes described in the following subsections change the default style.

Attributes Data

An attribute is added to the current style object if the data type opcode has value 0. This is the `gxStyleAttributesOpcode` constant.

The data, compressed as long, for the fields of the `gxStyleAttribute` structure follow and may be byte, short, or long.

The `gxStyleAttribute`s enumeration is described in the chapter "Geometric Styles" in *Inside Macintosh: QuickDraw GX Graphics.*

Tag Data

A tag is added to the current style if the data type opcode has value 1. This is the `gxStyleTagOpcode` constant. The data for the parameters of the `GXSetStyleTags` function follows.

The size of the opcode specifies the number of tags; the compression specifies whether the data is in bytes, words, or longs. For instance, if the size is 4 and the compression is `gxShortCompression` (2 bytes), then the stream contains 4/2 == 2 tags. The equivalent operation would be `GXSetShapeTags` (shape, nil, 1, 0, 2, tag array);

The `GXSetStyleTags` function is described in the chapter "Style Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Curve Error Data

A curve error, compressed as fixed-point, is added to the current style if the data type opcode has value 2. This is the `gxStyleCurveErrorOpcode` constant. The data for the `error` (`Fixed`) parameter of the `GXSetStyleCurveError` function follows.

For fixed point compression `gxNoCompression` means 1 `long` for each fixed number; `gxWordCompression` means 1 `short` for each fixed number treated as an integer (17 = 17.0); `gxByteCompression` means 1 byte per fixed number. Thus a byte compressed value can represent an integer fixed point number from –128.0 to 127.0; a word compression value can represent any integer fixed-point number.

The `GXSetStyleCurveError` function is described in the chapter "Geometric Styles" in *Inside Macintosh: QuickDraw GX Graphics.*

Pen Data

A pen, compressed as fixed point, is added to the current style object if the data type opcode has value 3. This is the `gxStylePenOpcode` constant. The data for the `pen` (`Fixed`) parameter of the `GXSetStylePen` function follows.

For fixed-point compression `gxNoCompression` means 1 `long` for each fixed number; `gxWordCompression` means 1 `short` for each fixed number treated as an integer (17 = 17.0); `gxByteCompression` means 1 byte per fixed number. Thus a byte compressed value can represent an integer fixed point number from –128.0 to 127.0; a word compression value can represent any integer fixed-point number.

The `GXSetStylePen` function is described in the chapter “Geometric Styles” in *Inside Macintosh: QuickDraw GX Objects.*

Join Data

A join is added to the current style object if the data type opcode has value 4. This is the `gxStyleJoinOpcode`. The data for the fields of the `gxJoinRecord` structure follows. The `gxJoinRecord` structure includes the `gxShape` and `gxJoinAttribute` structures.

The data sequence is omit byte, `attributes` (modifier flags) compressed as long, `join` (corner shape) compressed as long (reference), `miter` (size limit) compressed as fixed point.

The omit byte is interpreted by the `gxOmitJoinMask` and `gxOmitJoinShift` enumerations.

The `gxJoinAttribute` structure is described in the chapter “Geometric Styles” in *Inside Macintosh: QuickDraw GX Graphics.*

Dash Data

A dash is added to the current style object if the data type opcode has value 5. This is the `gxStyleDashOpcode` constant. The data for the fields of the `gxDashRecord` structure follows. The `gxShape` and `gxDashAttribute` enumerations are included in the `gxDashRecord` structure.

The data sequence is omit byte 1, `attributes` (modifier flags) compressed as long, `dash` (shape used for dash) compressed as long (reference), `advance` (distance between dashes) compressed as long, `phase` (start offset) compressed as fract, omit byte 2, and `scale` (height of dash) compressed as fixed.

In fract compression a long means a full fract; a word means that 16 bits are read followed by 16 bits of zeros; a byte means that 8 bits are read followed by 24 bits of zeros. Thus numbers like 1.0, -1.0, or fract 0.5 fit into a compressed byte.

Omit byte 1 is interpreted by the `gxOmitDashMask1` and `gxOmitDashShift1` enumerations. Omit byte 2 is interpreted by the `gxOmitDashMask2` and `gxOmitDashShift2` enumerations.

The `gxDashRecord` structure is described in the chapter “Geometric Styles” of *Inside Macintosh: QuickDraw GX Graphics.*

Caps Data

A cap is added to the current style object if the data type opcode has value 6. This is the `gxStyleCapsOpcode`. The data for the fields of the `gxCapRecord` structure follows. The `gxShape` and `gxCapAttribute` enumerations are included in the `gxCapRecord` structure.

The data sequence is omit byte, `attributes` (modifier flags) compressed as long, `startCap` (shape used at start of contours) compressed as long (reference), `endCap` (shape used at end of contours) compressed as long (reference).

The omit byte is interpreted by the `gxOmitCapMask` and `gxOmitCapShift` enumerations.

The `gxCapRecord` structure is described in the chapter "Geometric Styles" in *Inside Macintosh: QuickDraw GX Graphics.*

Pattern Data

A pattern is added to the current style object if the data type opcode has value 7. This is the `gxStylePatternOpcode` constant. The data for the fields of the `gxPatternRecord` structure follows. The `gxShape`, `gxPatternAttribute`, and `gxPoint` enumerations are included in the `gxPatternRecord` structure.

The data sequence is omit byte 1, `attributes` (modifier flags) compressed as long, `pattern` (shape to use as pattern) compressed as long (reference), `x` (x-coordinate of vector `u` for pattern grid) compressed as fixed, `y` (y-coordinate of vector `u` for pattern grid) compressed as fixed, omit byte 2, `x` (x coordinate of vector `v` for pattern grid) compressed as fixed, and `y` (y-coordinate of vector `v` for pattern grid) compressed as fixed. Note that for all of these, omit (zero) values are permitted.

Omit byte 1 is interpreted by the `gxOmitPatternMask1` and `gxOmitPatternShift1` enumerations. Omit byte 2 is interpreted by the `gxOmitPatternMask2` and `gxOmitPatternShift2` enumerations.

The `gxPatternRecord` structure is described in the chapter "Geometric Styles" in *Inside Macintosh: QuickDraw GX Graphics.*

Text Attributes Data

A text attribute compressed as long is added to the current style object if the data type opcode has value 8. This is the `gxStyleTextAttributesOpcode` constant. The data may be byte, word, or long.

The `gxTextAttribute` enumeration is described in the chapter "Typographic Styles" in *Inside Macintosh: QuickDraw GX Typography.*

Text Size Data

The text size, compressed as long, for the current style object is specified if the data type opcode has value 9. This is the `gxStyleTextSizeOpcode` constant. The data for the `size` (fixed point size of text) parameter of the `GXSetStyleTextSize` function follows.

The `GXSetStyleTextSize` function is described in the chapter “Typographic Styles” in *Inside Macintosh: QuickDraw GX Typography.*

Font Data

A font is added to the current style object if the data type opcode has value 10. This is the `gxStyleFontOpcode` constant. The attribute data for the `GXSetStyleFont` function follows. It is compressed as long (reference); the reference is to a font name defined earlier in the stream

The `GXSetStyleFont` function is described in the chapter “Typographic Styles” in *Inside Macintosh: QuickDraw GX Typography.*

Text Face Data

A text face is added to the current style object if the data type opcode has value 11. This is the `gxStyleTextFaceOpcode` constant. The data for the fields of the `gxTextFace` structure follows.

The data sequence is omit byte, `faceLayers` compressed as long, mapping size and `advanceMapping.`

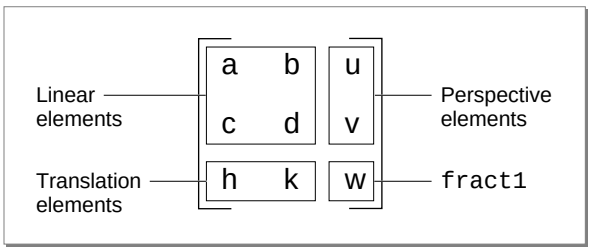
The `advanceMapping` in text face and transform mapping is reordered so that common mappings can be stored in fewer bytes. The omit byte and number of layers is followed by an optional byte (whose compression is described by `omitFaceMapping).`

The value of the byte may be one of the following:

Byte	Value
2	Mapping contains identity plus elements h and k.
4	Same as byte 2, plus elements a and d.
6	Same as byte 4, plus elements b and c.
9	Same as byte 6 plus elements u, v, and w.

The meaning of the elements mentioned in the previous table are shown in Figure 7-10.

Figure 7-10 Mapping matrix elements



The byte value is multiplied by the compression level to specify the length of the mapping data that follows. Byte compression multiplies by 1; word compression multiplies by 2; long compression multiplies by 4. The values in the left and middle columns are compressed as fixed values. The values in the right column are compressed as fract values. All elements whether the stream contains 2, 4, 6, or 9 numbers, have the same level of compression.

If the `faceLayers` value is greater than 0, then following the mapping data is an omit byte as described by `gxOmitFaceLayer` Mask 1. The omit byte is followed by the `outlineFill` compressed as a long, the flags comrpessed as a long, the `outlineStyle` and reference compressed as a long, and the `outlineTransform`, also comrpessed as a long. The second omit byte describes the bold x and bold y, compressed as fixed values. This sequence is repeated for the second and all remaining layers.

The omit byte is interpreted by the `gxOmitFaceMask` and `gxOmitFaceShift` enumerations.

The `gxTextFace` structure is described in the chapter “Typographic Styles” in *Inside Macintosh: QuickDraw GX Typography.*

Platform Data

The platform, script, and language is defined for the current object if the data type opcode has value 12. This is the `gxStylePlatformOpcode` constant. Data from the `gxFontPlatform`, `gxFontScript`, and `gxFontLanguage` enumerations follow.

The platform, script, and language are combined into a long and then that value is compressed as a long that is equal to

```
(platform << 16) | (script << 8) | language
```

The `gxFontPlatform`, `gxFontScript`, and `gxFontLanguage` enumerations are described in the chapter “Font Objects” in *Inside Macintosh: QuickDraw GX Typography.*

Font Variations Data

Font variations are added to the current style object if the data type opcode has value 13. The data is uncompressed. This is the `gxStyleFontVariationsOpcode` constant. The data for the fields of the `gxFontVariation` structure follows. The `gxFontVariationTag` structure is included in the `gxFontVariations` structure.

The data sequence is an array `[name` (variation tag), `value` (`Fixed`)`].` The opcode size specifies the number of variations in the stream.

The `gxFontVariation` structure is described in the chapter "Fonts" in *Inside Macintosh: QuickDraw GX Typography.*

Run Controls Data

Run controls are added to the current style object if the data type opcode has value 14. The data is uncompressed. This is the `gxStyleRunControlsOpcode` constant. The data for the fields of the `gxRunControls` structure follows. The opcode size specifies the size in bytes of the run control stream.

The `gxRunControls` structure is described in the chapter "Layout Line Controls" in *Inside Macintosh: QuickDraw GX Typography.*

Run Priority Justification Override Data

A run priority justification override is added to the current style object if the data type opcode has value 15. The data is uncompressed. This is the `gxStyleRunPriorityJustOverrideOpcode` constant. The data for the fields of the `gxPriorityJustificationOverride` structure follows. The opcode size specifies the size in bytes of the run control stream.

The data sequence is an array of delta. The opcode specifies the byte size.

The `gxPriorityJustificationOverride` structure is described in the chapter "Layout Line Controls" in *Inside Macintosh: QuickDraw GX Typography.*

Run Glyph Justification Overrides Data

A run glyph justification override is added to the current style object if the data type opcode has value 16. The data is uncompressed. This is the `gxStyleRunGlyphJustOverrideOpcode` constant. The data for the fields of the `gxGlyphJustificationOverride` structure follows. The `gxGlyphJustificationOverride` structure includes the `gxGlyphcode` and `gxWidthDeltaRecord` enumerations. The opcode specifies the byte size.

The data sequence is `count, glyphJustificationOverrides.`

The `gxGlyphJustificationOverride` structure is described in the chapter "Layout Line Controls" in *Inside Macintosh: QuickDraw GX Typography.*

Run Glyph Substitutions Data

A run glyph substitution is added to the current style object if the data type opcode has value 17. The data is uncompressed. This is the `gxStyleRunGlyphSubstitutionsOpcode` constant. The data for the fields of the `gxGlyphSubstitution` structure follows.

The data sequence is `count, glyphsubstitutions[].`

The `GXSetStyleRunGlyphSubstitutions` structure is described in the chapter "Layout Line Controls" in *Inside Macintosh: QuickDraw GX Typography.*

Run Features Data

A run feature is added to the current style object if the data type opcode has value 18. The data is uncompressed. This is the `gxStyleRunFeaturesOpcode` constant. The data for the fields of the `gxRunFeature` structure follows.

The data sequence is `count, runFeatures[].`

The `gxRunFeature` structure is described in the chapter "Layout Line Controls" in *Inside Macintosh: QuickDraw GX Typography.*

Run Kerning Adjustments Data

Run kerning adjustment is added to the current style object if the data type opcode has value 19. The data is uncompressed. This is the `gxStyleRunKerningAdjustmentsOpcode` constant. The data for the fields of the `gxKerningAdjustment` structure follows.

The data sequence is `count, kerningAdjustments[].`

The `gxKerningAdjustment` structure is described in the chapter "Layout Line Controls" in *Inside Macintosh: QuickDraw GX Typography.*

Style Justification Data

Style justification is added to the current style object if the data type opcode has value 20. The data is compressed as fract. This is the `gxStyleJustificationOpcode` constant. The data for the justify parameter of the `GXSetStyleJustification` function follows.

In fract compression a long means a full fract; a word means that 16 bits are read followed by 16 bits of zeros; a byte means that 8 bits are read followed by 24 bits of zeros. Thus numbers like 1.0, -1.0, or fract 0.5 fit into a compressed byte.

The `GXSetStyleJustification` function is described in the chapter "Typographic Styles" in *Inside Macintosh: QuickDraw GX Typography.*

New Ink Object Data

Data for a new ink object follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 29. This is the `gxInkTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration.

This opcode creates a new ink object with all of the properties of the previous ink object in the data stream. If the current ink object is the first ink object in the stream, then it is created with default properties. No data follows for the new ink object.

The ink object is described in the chapter "Ink Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Modified Ink Object Data

Once an ink object is defined, it can be modified by the addition of ink data. Modified style data follows a `gxSetDataOpcode` operation opcode and a data type opcode containing one of the constants from the `gxInkDataOpcode` enumeration. Table 7-6 summarizes the values of the constants for all of the modified ink objects.

Attributes Data

An attribute, compressed as long, is added to the current ink object if the data type opcode has value 0. This is the `gxInkAttributes`Opcode constant.

The data for the fields of the `gxInkAttributes` structure follow. The next two bytes contain the ink attribute flags.

The `gxInkAttributes` enumeration is described in the chapter "Ink Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Tag Data

A tag is added to the current ink object if the data type opcode has value 1. This is the `gxInkTagOpcode` constant. The data for the parameters of the `GXSetInkTags` function follows.

The size of the opcode specifies the number of tags; the compression specifies whether the data is in bytes, words, or longs. For instance, if the size is 4 and the compression is `gxShortCompression` (2 bytes), then the stream contains 4/2 == 2 tags. The equivalent operation would be `GXSetShapeTags` (shape, nil, 1, 0, 2, tag array).

The sequence is `tagType`, `index`, `oldCount`, `newCount`, `items`.

The `GXSetInkTags` function is described in the chapter "Ink Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Color Data

A color is added to the current ink object if the data type opcode has value 2. This is the `gxInkColorOpcode` constant. The data for the fields of the `gxInkAttributes` structure follow. The data for the fields of the `gxColor` structure follows.

The data sequence is omit byte, `space` (long), `profile` (long). The value of the omit byte may be omit compression.

The omit byte is interpreted by the `gxOmitColorsMask` and `gxOmitColorsShift` enumerations.

If space is indexed space, `gxOmitColoursIndex` is used to determine index compression (compressed as long), which is read first, followed by color set (compressed as long), with the compression determined by `gxOmitColorsIndexSet`.

If space is not indexed space, the color space determines the number of elements read from the stream as shown in Table 7-12.

Table 7-12 Color space and words read

16-bit	1
32-bit	2
gray, index	1
gray alpha	2
RGB, HSV, HLS, YXY, XYZ, LUV, LAB, YIQ	3
RGBA, CYMK	4

The bits in the omit byte determine whether a word is read from the stream for each word in the component or whether the byte is repeated twice for each word. For example, if the byte contains 0x3A, the word contains 0X3A3A. The `gxOmitColorsComponentsMask` sets 1 bit for up to 4 components.

The `gxColor` enumeration is described in the chapter “Ink Objects” in *Inside Macintosh: QuickDraw GX Objects.*

Transfer Mode Data

A transfer mode is added to the current ink object if the data type opcode has value 3. This is the `gxInkTransferModeOpcode` constant. The data for the fields of the `gxTransferMode` structure follow.

The data sequence is omit byte 1, `space`, compressed as long, `set`, compresssed as long, `profile`, compressed as long; omits are allowed. Omit byte 2 follows and then `sourceMatrix`, `deviceMatrix`, `resultMatrix`, `flags`, and `component`; omits are allowed.

The `sourceMatrix`, `deviceMatrix`, and `resultMatrix` are compressed as arrays of Fixed values. The color space determines the number of transfer components that follow, as shown in Table 7-12.

Each transfer component is preceded by an omit byte (`gxomitTransferComponentMask1`) that describes the first 4 fields of the structure. Omit byte one is followed by `gxOmitTransferComponentModeMask`, compressed as byte, `gxOmitTransferComponentFlagsMask`, compressed as byte, `gxOmitTransferComponentSourceMinimumShift`, compressed as color, `gxOmitTransferComponentSourceMaximumMask`, compressed as color, and `gxOmitTransferComponentDeviceMinimumMask`, compressed as color. Omit byte 2 follows which describes `gxOmitTransferComponentDeviceMaximumMask`, `gxOmitTransferComponentClampMinimumMask`, `gxOmitTransferComponentClampMaximumMask`, and `gxOmitTransferComponentOperandMask`; all these are compressed as color. The color compression specifies that the field may be omitted (inherits value from default), or is represented by a repeated byte (for example, `0X7A ==0X7A7A`), or is represented as a word.

Note that the mode and flags in the first omit byte have a single bit

The `gxTransferMode` structure is described in the chapter “Ink Objects” in *Inside Macintosh: QuickDraw GX Objects.*

New Object Transform Data

Data for a new transform object follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 0x2A. This is the `gxTransformTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration.

This opcode creates a new transform object with all of the properties of the previous transform object in the data stream. If the current transform object is the first transform object in the stream, then it is created with default properties. No data follows for the new transform object.

The transform object is described in the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects.* For additional information about transform mapping, see “Mapping Data” on page 7-46.

Modified Transform Object Data

Once a transform object is defined in the data stream, it can then be modified. Modified transform object data follows a `gxSetDataOpcode` operation opcode and a data type opcode containing one of the constants from the `gxTransformDataOpcode` enumeration. Table 7-9 summarizes the values of the constants for all of the modified transform objects.

Reserved Opcode for Modified Transform Data

The data type opcode with value 0 is reserved for future expansion.

Tag Data

A tag is added to the current transform object if the data type opcode has value 1. This is the `gxTransformTagOpcode` constant. The data for the parameters of the `GXSetTransformTags` function follows.

The data stream sequence is `tagType`, `index`, `oldCount`, `newCount`, `items[]`.

The `GXSetTransformTags` function is described in the chapter "Transform Objects" of *Inside Macintosh: QuickDraw GX Objects.*

Clip Data

A clip, compressed as long (reference) is added to the current transform object if the data type opcode has value 2. This is the `gxTransformClipOpcode` constant. The data for the `clip` parameter of the `GXSetTransformClip` function follows.

The `GXSetTransformClip` function is described in the chapter "Transform Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Mapping Data

A mapping is added to the current transform object if the data type opcode has value 3. This is the `gxTransformMappingOpcode` constant. The data for the `map` parameter of the `GXSetTransformMapping` function follows.

A transform mapping is initiated by the sequential appearance of the `gxSetDataOpcode`, and `gxTransformDataOpcode` constants in the data stream.

The bytes following the appearance in the data stream of the `gxTransformMapping` constant from the `gxTransformDataOpcode` enumeration have a special format. The interpretation of the bytes that follow require the determination of a size constant. The size to be used for a specific transform depends upon the compression and the size of the transform data specified by the byte containing the previous `gxGraphicsOperationOpcode` constant. The size is the number of bytes, words, or longs, depending upon the type of compression.

If the size obtained from the `gxGraphicsOperationOpcode` byte indicated that there are 24 bytes of transform data and the byte containing the `gxTransformMappingOpcode` constant indicated that there was no compression, then the size of each transform attribute would be 4 bytes (longs) and the size constant for our transformation bytes format would be size 24 / 4 = 6. The interpretation of the mapping that occurs for each mapping size is summarized in the section "Text Face Data" on page 7-39.

The `GXSetTransformMapping` function is described in the chapter "Transform Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Part Mask Data

A part mask, compressed as a long, is added to the current transform object if the data type opcode has value 4. This is the `gxTransformPartMaskOpcode` constant. The data for the `mask` parameter of the `GXSetTransformHitTest` function follows.

The `GXSetTransformHitTest` function is described in the chapter "Transform Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Tolerance Data

Tolerance, compressed as long, is added to the current transform object if the data type opcode has value 5. This is the `gxTransformToleranceOpcode` constant. The data for the tolerance parameter of the `GXSetTransformHitTest` function follows.

The `GXSetTransformHitTest` function is described in the chapter "Transform Objects" in *Inside Macintosh: QuickDraw GX Objects.*

New Color Profile Object Data

Data for a new color profile object follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 0x2B. This is the `gxColorProfileTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration.

This opcode creates a new color profile object with all of the properties of the previous color profile object in the data stream. If the current color profile object is the first color profile object in the stream, then it is created with default properties. The data that follows is uncompressed; the opcode size specifies the size of the stream.

The color profile object is described in the chapter "Color Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Modified Color Profile Object Data

Once a color profile object is defined in the data stream, it can be modified. Modified color set object data follows a `gxSetDataOpcode` operation opcode and a data type opcode containing one of the constants from the `gxColorProfileDataOpcode` enumeration. Table 7-8 summarizes the values of the constants for all of the modified color profile objects.

Reserved Opcode for Modified Color Profile Data

The data type opcode with value 0 is reserved for future expansion.

Color Profile Tag Data

A tag for the current color profile object is added if the data type opcode has value 1. This is the `gxColorProfileTagOpcode` constant. The data for the parameters of the `GXSetColorProfileTags` function follows.

The size of the opcode specifies the number of tags; the compression specifies whether the data is in bytes, words, or longs. For instance, if the size is 4 and the compression is `gxShortCompression` (2 bytes), then the stream contains 4/2 == 2 tags. The equivalent operation would be `GXSetShapeTags` (shape, nil, 1, 0, 2, tag array).

The `GXSetColorProfileTags` function is described in the chapter "Color Objects" in *Inside Macintosh: QuickDraw GX Objects.*

New Color Set Object Data

Data for a new color set object follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 0x2C. This is the `gxColorSetTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration.

This opcode creates a new color set object with all of the properties of the previous color set object in the data stream. If the current color set object is the first color set object in the stream, then it is created with default properties.

The color set object is described in the chapter "Color Objects" in *Inside Macintosh: QuickDraw GX Objects.*

Modified Color Set Object Data

Once a color set object is defined in the data stream, it can be modified. Modified color set object data follows an operation opcode `gxSetDataOpcode` constant from the `gxGraphicsOperationOpcode` enumeration and a data type opcode containing one of the constants from the `gxColorSetDataOpcode` enumeration. Table 7-7 summarizes the values of the constants for modified color set objects.

The first byte or two is space, space and specifies the number of components. The remaining stream is colors. The compression for the color set can be byte or word. To determine the number of colors in the stream use the following formula:

```
(size - colorSpaceByte * compression) / componentsInColorSpace *
                                                    compression
```

For instance, if the space is `gxRGBSpace`, the compression is `gxByteCompression`, and the size is 7, the number of colors would be (7 - 1 * 1)/3*1, which evaluates to 2. If the stream continued with 0, 0, 0, 0XFF, 0XFF, 0XFF, then the color set would contain black (0X0000, 0X0000,0X0000) and white 0XFFFF, 0XFFFF, 0XFFFF). As the example shows, the color set entries are compressed as colors. See section "Transfer Mode Data" on page 7-44 for information on color compression.

Reserved Opcode for Modified Color Set Data

The data type opcode with value 0 is reserved for future expansion.

Color Set Tag Data

A tag is added to the current color set object if the data type opcode has value 1. This is the `gxColorSetTagOpcode` constant. The data for the parameters of the `GXSetColorSetTags` function follows.

The size of the opcode specifies the number of tags; the compression specifies whether the data is in bytes, words, or longs. For instance, if the size is 4 and the compression is `gxShortCompression` (2 bytes), then the stream contains 4/2 == 2 tags. The equivalent operation would be `GXSetShapeTags` (shape, nil, 1, 0, 2, tag array).

The `GXSetColorSetTags` function is described in the chapter "Color Objects" of *Inside Macintosh: QuickDraw GX Objects.*

New Tag Object Data

Data for a new tag object follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 0x2D. This is the `gxTagTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration.

This opcode creates a new tag object with all of the properties of the previous tag object in the data stream. If the current tag object is the first tag object in the stream, then it is created with default properties. For tag data is uncompressed. The first parameter is tag type (long), followed by data computed from opcode length - sizeof (long).

The `GXNewtag` function is described in the chapter "Tag Objects" in *Inside Macintosh: QuickDraw GX Objects.*

New Bit Image Object Data

Data for a bit image object follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 0x2E. This is the `gxBitImageTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration.

The data sequence is omit byte (gxOmitBitImage), followed by the fields described by omit byte: `rowBytes`, compressed as long, `height`, compressed as long, and `data` compressed in the custom format described ahead. The bit image is compressed only if it makes the data stream smaller.

The `GXNewBitmap` function is described in the chapter "Bitmap Shapes" in *Inside Macintosh: QuickDraw GX Graphics.*

The bit image compression byte appears only in data streams containing a bitmap shape. This byte describes how each section of a bit image is compressed. The bit image compression byte follows the bytes containing the bit image attributes described by the `gxOmitBitImageMask` constant.

Bit images are described in the "Bitmap Shapes" chapter of *Inside Macintosh: QuickDraw GX Graphics.*

The bit image compression byte has the format *xx yyyyyy.*

The *xx* bits describe which of the bit image compression type opcodes is used for the next part of the bit image. The bit image compression opcode values are either 0, 1, 2, or 3.

The *yyyyyy* bits describe the number of times, z, that the action defined by the bit image compression opcode is replicated. The number of replications, *z*, can vary range from 0 to 63. Table 7-13 summarizes the four compression opcodes.

Table 7-13 Bit image compression opcodes

Bit image compression opcode	Bit image compression description
0	Add the *z* bytes of bit image that follow to the current row. *Z* Bytes of data follow.
1	Repeat 1 byte *z* times and add the bits to the current row. One byte of data follows.
2	Copy *z* bytes of the previous row and add the bits to the current row. No data follows.
3	Copy the previous row of bits *z* times and add the bits to the next z rows. No data follows.

The analysis of a bit image compression byte in a stream format is described in the section “Analyzing a Flattened Bitmap Shape” beginning on page 7-81.

New Font Name Data

Data for a font name follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 0x2F. This is the `gxFontNameTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration.

The fields in the `gxFlatFontName` structure follow. This structure includes the `gxFontName`, `gxFontPlatform`, `gxFontScript`, `gxFontLanguage`, and `gxFontName` structures, the byte length of the name and the name itself.

The stream exactly mirrors the sequence and size of the fields in the `gxFlatFontName` structure.

New Trailer Object Data

Data for a trailer object follows a `gxNewObjectOpcode` operation opcode and a data type opcode with a value 0x3F. This is the `gxTrailerTypeOpcode` constant from the `gxGraphicsNewOpcode` enumeration. This is the termination (last) object in the stream. No data follows.

The last two bytes of a stream are always 0x01 and 0x3F. The next to the last byte in a data stream contains a `gxNewObjectOpcode` constant with a record size of 1 byte. The last byte in a data stream contains a `gxTrailerTypeOpcode` constant with a `gxTwoBitCompression` value of 0, indicating the `gxNoCompression` constant.

About Print Files and Portable Digital Documents

QuickDraw GX printing performs background printing to all devices, allowing users continued access to the application. The printing process includes the creation of a specialized print file called a portable digital document.

Print Files

When an application prints, QuickDraw GX collects the printing information sent by the application and writes it to a file. This process is called *spooling* and the file that is created is called a **print file.** QuickDraw GX then reads the print file and prints it to the appropriate device. The read and interpretation process is called *despooling* and the printing process is called *imaging*.

A print file can be duplicated, dragged onto desktop printers, manipulated by print queues, and redirected to other printer devices without re-spooling. Print files also provide a device-independent information interchange format.

The QuickDraw GX spooling process consists of creating a print file and writing a stream of flattened shape data to that file. This data is unflattened during the unspooling process. Additional information must be provided in the print files. This includes job, formatting, and optimization information.

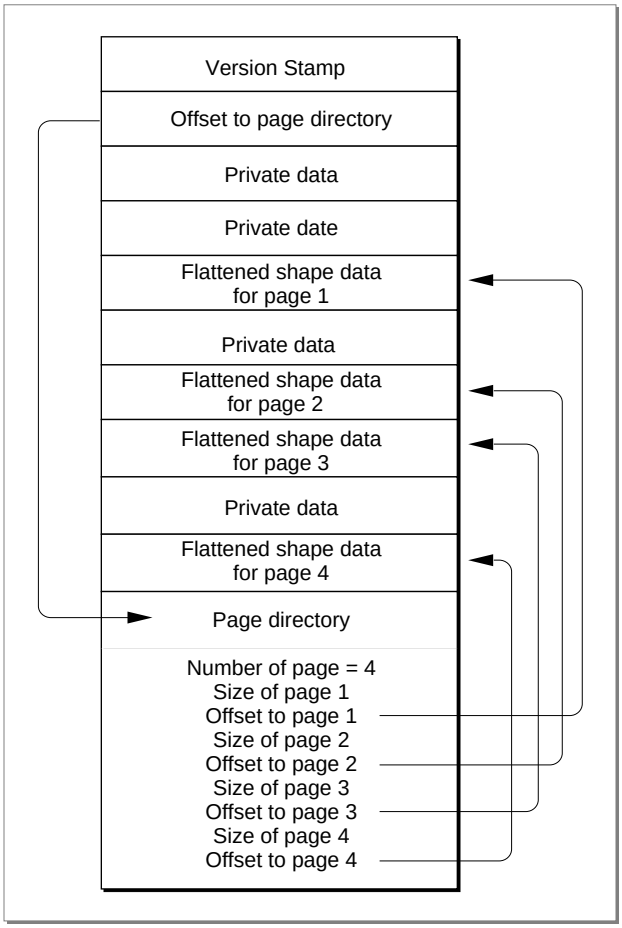
The job-related information includes the name of the job, the destination device, quality, and the number of copies. The formatting information includes the page sizes and orientations. The optimization information includes the font database.

The print file consists of two forks, a data fork and a resource fork. The data fork contains all the core data necessary to print a document. This consists of the flattened job data, the flattened shape data for each page, and the flattened format data for each page.

The print file begins with a 32-bit QuickDraw GX version followed by a 32-bit offset that describes the number of bytes from the beginning of the file to the start of the page directory located at the end of the file.

The page directory contains a 32-bit number indicating the number of pages in the document, an array of page sizes, and offsets to the start of the flattened shape data for each page. The format of a print file for a four-page document is shown in Figure 7-11.

Figure 7-11 Print file format



QuickDraw Picture Data in Print Files

When creating a print file from a document that contains QuickDraw drawing commands, QuickDraw GX by default saves the QuickDraw data for each page in a tag object of tag type `'pict'` attached to a rectangle shape. Therefore, if you are examining the data stream of a print file, you should note that a rectangle shape with an attached tag object of type `'pict'` indicates the presence of QuickDraw data. For more information about this tag object and QuickDraw data, see the discussion of the `'pict'` tag object in the advanced printing features chapter of *Inside Macintosh: QuickDraw GX Printing.* ◆

Portable Digital Documents

QuickDraw GX provides document portability that is independent of fonts, applications, and output devices. The users of your application can create and save their results in the form of a **portable digital document** or **PDD.**

A portable digital document consists of the print file containing flattened shapes described in the previous section. These files provide all of the information necessary to view and print the document, including the fonts that are used and other information necessary to render the text and graphics. A portable digital document can be sent to other Macintosh users and viewed or printed simply by opening the documents with a viewer that can interpret them.

For more information on print files and portable digital documents, see the chapters “Introduction to QuickDraw GX Printing” and “Core Printing Features” of *Inside Macintosh: QuickDraw GX Printing.*

Using QuickDraw GX Stream Format

This section describes the use of the GraphicsBug utility to analyze flattened data streams. Sample code is provided that draws a QuickDraw GX picture containing seven shapes. GraphicsBug is used to flatten each shape. The resulting data stream for each flattened shape is then analyzed.

This section describes how you can

- flatten shapes using GraphicsBug
- interpret the GraphicsBug flattened shape output format
- analyze flattened shape data streams

Flattening Shapes With GraphicsBug

GraphicsBug is not just a QuickDraw GX debugging tool. It also allows you to evaluate the data at specific memory locations. You can use GraphicsBug to look at the data describing a QuickDraw GX shape both before and after you invoke the `GXFlattenShape` function. This allows you to compare the original data and the stream format after the `GXFlattenShape` function has been called.

For more information concerning GraphicsBug, see the chapter “QuickDraw GX Debugging ” in this book.

You can use GraphicsBug to analyze a data stream by using the following procedure:

1. Create a QuickDraw GX shape.
2. Use the GraphicsBug heap dump `HD` command to determine the memory location of the QuickDraw GX shape to be flattened.
3. Copy the memory location of the shape to the clipboard.
4. Type `FL` and paste the memory address. The command line should look like this:

```
fl <memory address>
```

For example: `fl 41d788`

5. The command `FL` applies `the GXFlattenShape` function to the shape located at the specified memory address. This results in a flattened shape. An annotated version of the QuickDraw GX data stream appears in the GraphicsBug window. GraphicsBug does not alter the graphics memory in any way.

To create a flattened file, you can use the command

```
fl <memory address> "filename"
```

To view the contents of a file, such as a print file generated by printing a document, you can use the command

```
uf "filename"
```

To view the stream associated with a particular page of a document, you can use the command

```
uf <page number> "filename"
```

Here are some guidelines for using GraphicsBug to analyze data streams:

- The data in parentheses in the GraphicsBug window are the compressed byte codes that were generated when the original shape was flattened. The data not in parenthesis is GraphicsBug's brief annotation of the data stream. The annotation usually describes the shape data in its original format. The data in parentheses always relates to the immediately previous data that is not in parentheses.
- Sometimes GraphicsBug will not give the name of the font. This is because GraphicsBug reads only the information contained in memory. GraphicsBug cannot make a call to get the information. If GraphicsBug is used to flatten shapes that were generated by a client call, the required data will always already be in memory and will therefore be available. In this case, the GraphicsBug annotation will always provide the name of the font.
- If part of an object is compressed and another part of the object is not compressed, GraphicsBug reports that there is "no compression."
- Bracketed numbers are references. When `gxSetData` or `gxSetReference` opcodes are encountered, they can't generate pointers to other objects. They have to generate references. The first object is given reference 1. Subsequent objects are given references 2, 3, and so on.

Listing 7-2 shows an example of the information provided by GraphicsBug for a flattened line.

Listing 7-2 A GraphicsBug annotation of the data stream of a flattened shape

```
fl 0c79090
owners            1)
newObject; size: #2 (03)
headerType; byte compression (80)
version == 1.0; flags == fontListFlatten | fontGlyphsFlatten
(01 03)
newObject; size: #6 (07) [1]
fontNameType; no compression (2f)
(04 02 01 01 00 00)
```

Listing 7-2 shows only the beginning of a data stream. For more examples of GraphicsBug annotation of flattened shape data streams, see the next section.

Analyzing the Data Streams of Flattened Shapes

This section first uses sample code to generate a picture with seven shapes. Each of the seven shapes is then flattened using the procedure described in the section "Flattening Shapes With GraphicsBug" beginning on page 7-54. The section "Analyzing the Data Streams of Flattened Shapes" beginning on page 7-56 describes how to use GraphicsBug to interpret the data for each of the seven shapes. The GraphicsBug data stream output is provided for each flattened shape in Listing 7-4 through Listing 7-10. The byte-by-byte analysis of the data stream for each flattened shape is provided in Table 7-14 through Table 7-20.

Creating a Picture With Seven Shapes

Listing 7-3 creates seven primitive shapes and adds them to a window's page shape to form the picture shown in Figure 7-12. This picture contains (from left to right and top to bottom) a line, rectangle, curve, path, text, polygon and bitmap shape.

Listing 7-3 A picture with seven shapes

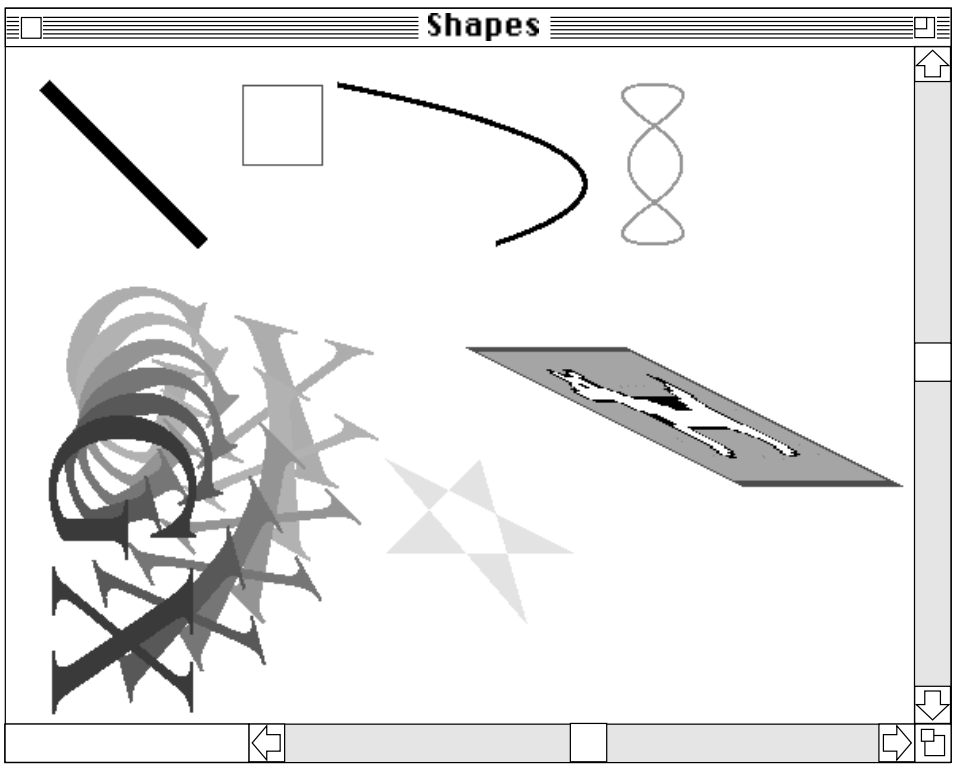
```
void CreateSampleImage(WindowPtr wind)
{
gxShape thePage;
gxShape theLine;
line lineData = {ff(25), ff(25), ff(125), ff(125)};
gxShape theRect;
gxRectangle rectData = {ff(25), ff(25), ff(75), ff(75)};
gxShape theCurve;
gxCurve curveData = {ff(25), ff(25), ff(275), ff(75), ff(125),
ff(125)};
gxShape thePath;
long tripleEightData[] = {1/* # of contours */, 6 /* # of points
*/, 0xff000000,
   0, 0,
   ff(75),  0,
   ff(5), ff(50),
   ff(75), ff(100),
   0, ff(100),
   ff(75), ff(50)};
gxShape theText;
gxRectangle theTextBounds;
gxColor textColor;
fixed x,y;
short loop;
gxShape thePolygon;
```

```
long starData[] = {1,  /* number of contours */ 5, /* number of
points */
   ff(60), 0, ff(90), ff(90),  ff(0), ff(30),  ff(120), ff(30),
   ff(0), ff(90)}; /* the points */
   gxShape theBitmap;

/* retrieve the page shape so we can add to it */
thePage = GetDocShape(wind);

/* Create a line shape*/

   theLine = GXNewLine (&lineData);
   GXSetShapePen(theLine, ff(9));
   GXAddToShape(thePage, theLine);
   GXDisposeShape(theLine);

/* create a rectangle; the color of the rectangle is red */

theRect = GXNewRectangle(&rectData);
   {gxColor redColor =
      {gxRGBSpace, nil,{
         0xFFFF,0,0}};
GXSetShapeColor(theRect, &redColor);
   }
GXSetShapeFill (theRect, closedFrameFill);
GXMoveShapeTo (theRect,  ff(150), ff(25));
GXAddToShape(thePage, theRect);
GXDisposeShape(theRect);

/* create a curve shape; the shape has a pen thickness of 3.25 */

theCurve = GXNewCurve(&curveData);
GXSetShapePen(theCurve, fl(3.25));
GXMoveShapeTo (theCurve,  ff(210), ff(25));
GXAddToShape(thePage, theCurve);
GXDisposeShape(theCurve);

/* create a path shape; the shape's color is green and the pen
thickness is 2 */

thePath = GXNewPaths((paths *) tripleEightData);
GXSetShapeFill (thePath, closedFrameFill);
   GXSetShapePen(thePath, ff(2));
GXSetShapeCommonColor (thePath, green);
```

```
GXMoveShapeTo (thePath,  ff(390), ff(25));
GXAddToShape(thePage, thePath);
GXDisposeShape(thePath);

/* create a text shape; the shape is the characters GX colored in
hsv space and rotated 90 degrees */

/* create the text, set the font size, and set the font name */

theText = NewText(2,(unsigned char*)"GX",  nil);
GXSetShapeCommonFont(theText, timesFont);
GXSetShapeTextSize(theText, ff(135));
GXMoveShapeTo (theText,  ff(25), ff(230));
GXSetShapeAttributes (theText,  gxMapTransformShape);

/* create an hsv color space and set up the initial colors */

textColor.space = hsvSpace;
textColor.profile = nil;
textColor.element.hsv.hue = 0x7400;
textColor.element.hsv.saturation = 0xFFFF;
textColor.element.hsv.value = 0xFFFF;

/* get the bounds of "theText" and determine the coordinates of
the bottom left corner */
GXGetShapeBounds(theText, 0L, &theTextBounds);
x = theTextBounds.left;
y = theTextBounds.bottom;

/* rotate "theText"; add each letter to the picture */
for (loop = 0; loop < 6; loop++) {
   GXSetShapeColor(theText, &textColor);
   GXRotateShape(theText, ff(90), x, y);
   GXAddToShape(thePage, theText);
   textColor.element.hsv.hue += 0x0940;
}
GXDisposeShape(theText);

/* create a polygon shape; the shape's color is yellow, the pen
size is 3, and it is skewed in the vertical direction by a factor
of 0.5 */

thePolygon = GXNewPolygons((gxPolygons *) starData);
GXSetShapeFill(thePolygon, gxEvenOddFill);
```

```
GXSetShapePen (thePolygon, ff(3));
GXSetShapeCommonColor (thePolygon, yellow);
GXMoveShapeTo (thePolygon,  ff(240), ff(110));
GXSkewShape(thePolygon, 0, fl(0.5), 0, 0);

GXAddToShape(thePage, thePolygon);
GXDisposeShape(thePolygon);

/* create a bitmap by retrieving a bitmap from the resource fork
and skewing it in the horizontal direction by a factor of .*/

   theBitmap = GXGetPixMapShape(128);
   GXValidateShape (theBitmap);

   GXSkewShape(theBitmap, ff(2), 0, 0, 0);
   GXMoveShapeTo (theBitmap,  ff(290), ff(190));

   GXAddToShape(thePage, theBitmap);
   GXDisposeShape(theBitmap);
```

Figure 7-12 A picture with seven shapes

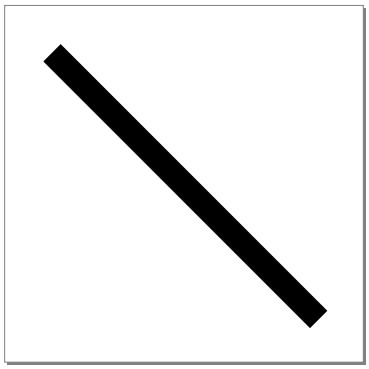


Analyzing a Flattened Line Shape

The function described in the section “Creating a Picture With Seven Shapes” beginning on page 7-56 was first used to draw the picture shown in Figure 7-12 containing the line shape shown in Figure 7-13.

The line shape is created with a pen size of 9 and a default color of black. The pen is moved from the point (25.0, 25.0) to point (125.0, 125.0).

Figure 7-13 The line shape drawn



The procedure described in the section “Flattening Shapes With GraphicsBug” beginning on page 7-54 was then used to generate the GraphicsBug output shown in Listing 7-4. The first line of the output shows the use of the `fl` command on the memory address that contained the line shape. The flattened line shape data stream is the sequential byte data that appears in parentheses. For example, the first four bytes of the data stream in Listing 7-4 are (06) (80) (01 03). All other annotation is provided by GraphicsBug.

Since the flattened line shape is the first shape in the data stream, this first part of the GraphicsBug output shows the data stream header. The GraphicsBug output for the other flattened shapes described in this section correspond to the data stream that describes that specific shape. These shape-specific sections are presented in QuickDraw GX drawing order.

Listing 7-4 GraphicsBug analysis of a flattened line

```
fl 0c79090
owners            1)
newObject; size: #2 (03)
headerType; byte compression (80)
version == 1.0; flags == fontListFlatten | fontGlyphsFlatten
(01 03)
newObject; size: #6 (07) [1]
fontNameType; no compression (2f)
```

```
(04 02 01 01 00 00)
newObject; size: #0 (01) [1]
styleType; no compression (28)
setData; size: #1 (42)
stylePen; byte compression (83)
(09)
newObject; size: #0 (01) [1]
inkType; no compression (29)
newObject; size: #0 (01) [1]
transformType; no compression (2a)
newObject; size: #4 (05)
lineType; byte compression (83)
(19 19 7d 7d)
newObject; size: #0 (01)
trailerType; no compression (3f)
```

Table 7-14 shows the data stream analysis of the flattened line shape. The stream data is obtained from the GraphicsBug output in Listing 7-4. This table provides a description of each byte of the data stream for this shape.

Table 7-14 Analysis of the data stream of a flattened line shape

Values in data stream (binary)	Type of information	Value	Description
New header			
0x03	Operation opcode	0	New object
(00 000011)	Record size	3	Record size is 3 bytes
0x80	Compression type opcode	2	Byte compression
(10 000000)	Data type opcode	0	Header
0x01 (00000001)	Data	1.0	QuickDraw GX Version 1.0
0x03 (00000011)	Data	3	`gxFontListFlatten` constant from the `gxFlattenFlags` enumeration is 0x01 `gxFontGlyphsFlatten` constant from the `gxFlattenFlags` enumeration is 0x02

continued

Table 7-14 Analysis of the data stream of a flattened line shape (continued)

Values in data stream (binary)	Type of information	Value	Description
New font name for the style object			
0x07 (00 000111)	Operation opcode	0	New object
	Record size	7	Record size is 7 bytes
0x2F (00 101111)	Compression type opcode	2	No compression
	Data type opcode	0x2F	Font name
0x04	Data	4	The `gxUniqueFontName` constant of the `gxFontName` enumeration
0x02	Data	2	The `gxMacintoshPlatform` constant of the `gxFontPlatform` enumeration
0x01	Data	1	The `gxMacintoshRomanScript` constant of the `gxMacintoshScripts` enumeration
0x01	Data	1	The `gxEnglishLanguage` constant of the `gxFontLanguage` enumeration
0x0001A	Data	26	The `length` field (short) of the `gxFontName` is 26 bytes.
0x41 70 70 6C 65 20 43 6F 6D 70 75 74 65 72 20 54 69 6D 65 73 20 52 6F 6D 61 6E			
	Data		Each of the 26 bytes is one glyph code. The font name is “Apple Computer Times ROman.”
New style object			
0x01 (00 000001)	Operation opcode	0	New object
	Record size	1	Record size is 1 byte
0x28 (00 101000)	Compression type opcode	2	No compression
	Data type opcode	0x28	New style
0x42 (01 000010)	Operation opcode	1	Set data
	Record size	1	Record size is 1 byte

Table 7-14 Analysis of the data stream of a flattened line shape (continued)

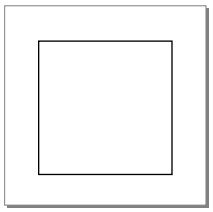
Values in data stream (binary)	Type of information	Value	Description
0x83 (10 000011)	Compression type opcode	2	No compression
	Data type opcode	3	`gxStylePenOpcode` constant of the `gxStyleDataOpcode` enumeration
0x09	Data	9.0	The pen width parameter for the `GXSetShapePen` function is 9.0
New ink object			
0x01 (00 000001)	Operation opcode	0	New object
	Record size	1	Record size is 1 byte
0x29 (00 101001)	Compression type opcode	0	No compression
	Data type opcode	0x29	New ink
New transform			
0x01 (00 000001)	Operation opcode	0	New object
	Record size	1	Record size is 1 byte
0x2A (00 101001)	Compression type opcode	0	No compression
	Data type opcode	0x2A	New transform
New shape object			
0x05 (00 000101)	Operation opcode	0	New object
	Record size	5	Record size is 5 bytes
0x83 (10 000011)	Compression type opcode	2	Byte compression
	Data type opcode	3	`gxLineType` constant of the `gxShapeTypes` enumeration
0x19	Data	25.0	x coordinate of the first point is 25.0
0x19	Data	25.0	y coordinate of the first point is 25.0
0x7D	Data	125.0	x coordinate of the last point is 125.0
0x7D	Data	125.0	y coordinate of the last point is 125.0

Analyzing a Flattened Rectangle Shape

The function described in section “Creating a Picture With Seven Shapes” beginning on page 7-56 was first used to draw the picture shown in Figure 7-12 containing the rectangle shape shown in Figure 7-14.

The red rectangle shape is created with its frame. The size and shape of the rectangle is defined by its upper-left boundary point (25.0, 25.0) and its lower-right boundary point (75.0, 75.0). The fill type is closed-frame. Once the rectangle is drawn, it is moved to the point (150.0, 25.0) to position it in the picture.

Figure 7-14 The rectangle shape drawn



The procedure described in the section “Flattening Shapes With GraphicsBug” beginning on page 7-54 was then used to generate the GraphicsBug output shown in Listing 7-5. The flattened rectangle shape data stream is the sequential data that appears in parentheses.

Listing 7-5 GraphicsBug analysis of a flattened rectangle shape

```
inkType; no compression (29)
  space        gxRGBSpace
  profile           nil
  value(s)       1.0000 (ffff)  0.0000 0x0000  0.0000 0x0000
setData; size: #4 (45)
inkColor; no compression (02)
(fe ff 00 00)
newObject; size: #8 (09)
rectangleType; word compression (45)
(00 96 00 19 00 c8 00 4b)
setData; size: #1 (42)
shapeFill; byte compression (82)
(02)
```

Table 7-15 shows the data stream analysis of the flattened rectangle shape. The stream data is obtained from the GraphicsBug output in Listing 7-5. This table provides a description of each byte of the data stream for this shape. Data format sequences that are identical to previously described data sequences in the stream are not shown..

Table 7-15 Analysis of the data stream of a flattened rectangle shape

Values in data stream (binary)	Type of information	Value	Description
New ink object			
0x01 (00 000001)	Operation opcode	0	New object
	Record size	1	Record size is 1 byte.
0x29 (00 101001)	Compression type opcode	0	No compression
	Data type opcode	0x29	New ink
Set data for ink color			
0x45 (01 000101)	Operation opcode	1	Set data
	Record size	5	Record size is 5 bytes.
0x02 (00 000010)	Compression type opcode	0	No compression
	Data type opcode	2	`gxInkColorOpcode` constant of the `gxInkDataOpcode` enumeration
0xFE (11 11 1110)	Omit byte	–	The `gxOmitColorsMask` and `gxOmitColorsShift` enumerations are used to interpret this byte. Data1, color space, is omitted so the default RGB color space properties are applied to the current object. Data2, color profile, is omitted so the default color profile is applied to the current object. Data3, color components, uses only bits 3, 2, and 1 for RGB. The compression for each of the red, green, and blue color components is byte compression.
0xFF	Data	0xFFFF	Since color components are 2-byte values, the byte is replicated to the value 0xFFFF or 65,535. The RGB value for the `red` field of the `gxRgbColor` structure is 65,535.

continued

Table 7-15 Analysis of the data stream of a flattened rectangle shape (continued)

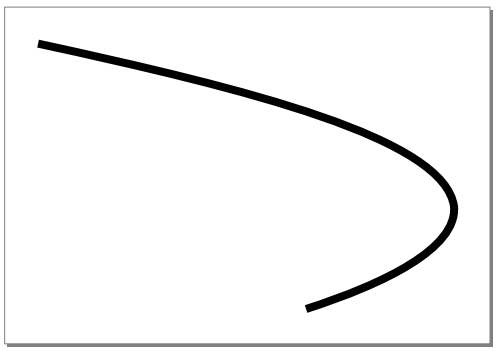
Values in data stream (binary)	Type of information	Value	Description
0x00	Data	0x0000	Since color components are 2-byte values, the byte is replicated to the value 0x0000 or 0. The RGB value for the `green` field of the `gxRgbColor` structure is 0.
0x00	Data	0x0000	Since color components are 2-byte values, the byte is replicated to the value 0x0000 or 0. The RGB value for the `blue` field of the `gxRgbColor` structure is 0.
New rectangle object			
0x09 (00 001001)	Operation opcode	0	New object
	Record size	5	Record size is 9 bytes.
0x45 (01 000101)	Compression type opcode	1	Word compression
	Data type opcode	5	`gxRectangleType` constant of the `gxShapeTypes` enumeration
0x00 96	Data	150.0	x-coordinate of the left top corner point is 150.0
0x00 19	Data	25.0	y-coordinate of the left top corner point is 25.0
0x00 C8	Data	200.0	x-coordinate of the right bottom corner point is 200.0
0x00 4B	Data	125.0	y-coordinate of the right bottom corner point is 75.0
Set data for shape fill			
0x42 (01 000010)	Operation opcode	1	Set data
	Record size	2	Record size is 2 bytes.
0x82 (10 000010)	Compression type opcode	2	Byte compression
	Data type opcode	2	`gxShapeFillOpcode` constant of the `gxShapeDataOpcode` enumeration
0x02	Data	2	`gxClosedFrameFill` constant of the `gxShapeFills` enumeration. The shape fill constant is a long number so the byte is expanded to a long.

Analyzing a Flattened Curve Shape

The function described in the section “Creating a Picture With Seven Shapes” beginning on page 7-56 was first used to draw the picture shown in Figure 7-12 containing the curve shape shown in Figure 7-15.

The curve has a pen thickness of 3.25. The size and shape of the curve are defined by its first point (210.0), control point (460.0, 75.0), and last point (310.0, 125.0). Once the curve is drawn, it is moved to the point (210.0, 25.0) to position it in the picture.

Figure 7-15 The curve shape drawn



The procedure described in the section “Flattening Shapes With GraphicsBug” beginning on page 7-54 was then used to generate the GraphicsBug output shown in Listing 7-6. The flattened curve shape data stream is the sequential data that appears in parentheses.

Listing 7-6 GraphicsBug analysis of a flattened curve shape

```
.
.
.
newObject; size: #6 (07) [1]
fontNameType; no compression (2f)
(04 02 01 01 00 00)
newObject; size: #0 (01) [1]
styleType; no compression (28)
setData; size: #4 (45)
stylePen; no compression (03)
(00 03 40 00)
.
.
.
```

```
newObject; size: #12 (0d)
curveType; word compression (44)
(00 d2 00 19 01 cc 00 4b 01 36 00 7d)
newObject; size: #0 (01)
trailerType; no compression (3f)
```

Table 7-16 shows the data stream analysis of the flattened rectangle shape. The stream data is obtained from the GraphicsBug output in Listing 7-6. This table provides a description of each byte of the data stream for this shape. Data format sequences that are identical to previously described data sequences in the stream are not shown and are not analyzed here.

Table 7-16 Analysis of the data stream of a flattened curve shape

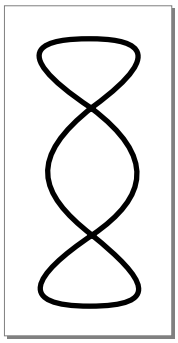
Values in data stream (binary)	Type of information	Value	Description
0x45	Operation opcode	1	Set data.
(01 000101)	Record size	5	Record size is 5 bytes.
0x03	Compression type opcode	0	No compression
(00 000011)	Data type opcode	3	`gxStylePenOpcode` constant of the `gxStyleDataOpcode` enumeration
0x00034000	Data	3.25	The pen width parameter for the `GXSetPen` function is 3.25.
.			
.			
.			
0x0D	Operation opcode	0	New object
(00 01101)	Record size	13	Record size is 13 bytes.
0x44	Compression type opcode	1	Word compression
(01 000100)	Data type opcode	4	`gxCurveType` constant of the `gxShapeTypes` enumeration
0x00 D2	Data	210.0	x-coordinate of the first point is 210.0.
0x00 19	Data	25.0	y-coordinate of the first point is 25.0.
0x00 CC	Data	460.0	x-coordinate of the control point is 460.0.
0x00 4B	Data	75.0	y-coordinate of the control point is 75.0.
0x00 36	Data	310.0	x-coordinate of the last point is 310.0.
0x00 7D	Data	125.0	x-coordinate of the last point is 125.0.

Analyzing a Flattened Path Shape

The function described in the section “Creating a Picture With Seven Shapes” beginning on page 7-56 was first used to draw the picture shown in Figure 7-12 containing the path shape shown in Figure 7-16.

A path is created with a pen thickness of 2.0 and a color of green. The size and shape of the curve are defined by the points (0.0, 0.0), (75.0, 0.0), (5.0, 50.0), (75.0, 100.0), (0.0, 100.0), and (75.0, 50.0). Once the path is drawn, it is moved to the point (290.0, 25.0) to position it in the picture. The line is not on any of the points.

Figure 7-16 The path shape drawn



The procedure described in the section “Flattening Shapes With GraphicsBug” beginning on page 7-54 was then used to generate the GraphicsBug output shown in Listing 7-7. The flattened path shape data stream is the sequential data that appears in parentheses.

Listing 7-7 GraphicsBug analysis of a flattened path shape

```
newObject; size: #0 (01) [1]
transformType; no compression (2a)
newObject; size: #19 (14)
pathType; byte compression (87)
(01 06 ff 2a 01 73 40 00 19 b5 00 46 ce ba ce 4b 00 b5 32)
setData; size: #1 (42)
shapeFill; byte compression (82)
(02)
```

Table 7-17 shows the data stream analysis of the flattened path shape. The stream data is obtained from the GraphicsBug output in Listing 7-7. This table provides a description of each byte of the data stream for this shape. Data format sequences that are identical to previously described data sequences in the stream are not shown and are not analyzed here.

Table 7-17 Analysis of the data stream of a flattened path shape

Values in data stream (binary)	Type of information	Value	Description
New path object			
`0x14 (00 010100)`	Operation opcode	0	New object
	Record size	14	Record size is 14 bytes.
`0x87 (10 000111)`	Compression type opcode	2	Byte compression
	Data type opcode	7	`gxPathType` constant of the `gxShapeTypes` enumeration
`0x01`	Data	1	The number of contours is 1.
`0x06`	Data	6	The number of points in the contour is 6.
`0xFF (111111 11)`	Control byte	–	Each of the 6 points is assigned a control bit from the control byte. Points having a 0 bit are on the line. Points having a 1 bit are off the line. All 6 points are off the line. The final 2 bits are unused.
`0x2A (00 10 10 10)`	Omit byte	–	The `gxOmitPathMask` and `gxOmitPathShift` enumerations are used to interpret this byte. No compression is used for data1, x coordinate of first point. Byte compression is used for data2, y coordinate of first point. Byte compression is used for data3, all x relative coordinate deltas. Byte compression is used for data4, all y relative coordinate deltas.
`0x01734000`	Data1	371.25	Absolute x-coordinate of the first point is 371.25.

Table 7-17 Analysis of the data stream of a flattened path shape (continued)

Values in data stream (binary)	Type of information	Value	Description
0x0x19	Data2	25.0	Absolute y-coordinate of the first point is 25.0.
0xB5	Data3	-75.0	Relative x-coordinate of the second point is -75.0. Absolute x coordinate is 371.25 – (–75.0) = 446.25.
0x00	Data4	0.0	Relative y-coordinate of the second point is 0. Absolute y coordinate is 25.0 – (0.0) = 25.0.
0x46	Data3	70.0	Relative x-coordinate of the third point is 70.0. Absolute x coordinate is 371.25 – (70.0) = 301.25.
0xCE	Data4	–50.0	Relative y-coordinate of the third point is –50.0. Absolute y coordinate is 25.0 – (–50.0) = 75.0.
0xBA	Data3	–70.0	Relative x coordinate of the fourth point is -70.0. Absolute x-coordinate is 371.25 – (–70.0) = 441.25.
0xCE	Data4	–50.0	Relative y coordinate of the fourth point –50.0. Absolute y-coordinate is 25.0 – (–50.0) = 75.0.
0x4B	Data3	75.0	Relative x coordinate of the fifth point is 75.0. Absolute x-coordinate is 371.25 – (75.0) = 296.25.
0x00	Data4	0.0	Relative y coordinate of the fifth point is 0.0. Absolute y-coordinate is 25.0 – (0.0) = 25.0.
0xB5	Data3	–75.0	Relative x coordinate of the sixth point is –75.0. Absolute x-coordinate is 371.25 – (–75.0) = 446.25.
0x32	Data4	50.0	Relative y coordinate of the sixth point is 50.0. Absolute y-coordinate is 25.0 – (50.0) = –25.0.

Analyzing a Flattened Text Shape

The function described in the section “Creating a Picture With Seven Shapes” beginning on page 7-56 was first used to draw the picture shown in Figure 7-12 containing the path shape shown in Figure 7-17.

A text shape with glyphs G and X is colored in hsv space. The glyphs are rotated six times by 90 degrees about the left bottom corner. Once the text is drawn, it is moved to the point (25.0, 230.0) to position it in the picture.

Figure 7-17 The text shape drawn

The procedure described in the section “Flattening Shapes With GraphicsBug” beginning on page 7-54 was then used to generate the GraphicsBug output shown in Listing 7-8. The flattened text shape data stream is the sequential data that appears in parentheses.

Listing 7-8 GraphicsBug analysis of a flattened text shape

```
newObject; size: #32 (21) [1]
fontNameType; no compression (2f)
(04 02 01 01 00 1a)
Apple Computer Times Roman
(41 70 70 6c 65 20 43 6f 6d 70 75 74 65 72 20 54 69 6d 65 73 20
52 6f 6d 61 6e)
newObject; size: #0 (01) [1]
styleType; no compression (28)
setData; size: #1 (42)
styleFont; byte compression (8a)
(01)
setData; size: #2 (43)
styleTextSize; word compression (49)
(00 87)
newObject; size: #0 (01) [1]
inkType; no compression (29)
  space        hsvSpace
  profile           nil
  value(s)       0.4531 0x7400  1.0000 (ffff)  1.0000 (ffff)
setData; size: #6 (47)
inkColor; no compression (02)
(b6 03 74 00 ff ff)
newObject; size: #0 (01) [1]
transformType; no compression (2a)
setData; size: #24 (59)
transformMapping; no compression (03)
(00 3d 02 12 00 00 98 fe 00 00 f7 47 00 00 f7 47 00 00 42 42 ff
ff bd be)
newObject; size: #8 (09)
textType; no compression (09)
  byteLength        2
  position                 {    25.0000,   230.0000}
Displaying memory from 00c7a116
 00c7a116  4758                                      GX
(a4)
bytes (02)
position.x (19)
position.y (00 e6 02 47 58)
setData; size: #1 (42)
shapeAttributes; byte compression (80)
(20)
```

Table 7-18 shows the data stream analysis of the flattened rectangle shape. The stream data is obtained from the GraphicsBug output in Listing 7-8. This table provides a description of each byte of the data stream for this shape. Data format sequences that are identical to previously described data sequences in the stream are not shown and are not analyzed here.

Table 7-18 Analysis of the data stream of a flattened text shape

Values in data stream (binary)	Type of information	Value	Description
New font name for the style object			
0x21 (00 100001)	Operation opcode	0	New object
	Record size	21	Record size is 21 bytes
0x2F (0010111)	Compression type opcode	0	No compression
	Data type opcode	7	`gxFontNameOpcode` constant of the `gxGraphicsNewOpcode` enumeration
0x04	Data	4	The `gxUniqueFontName` constant of the `gxFontName` enumeration
0x02	Data	2	The `gxMacintoshPlatform` constant of the `gxFontPlatform` enumeration
0x01	Data	1	The `gxMacintoshRomanScript` constant of the `gxMacintoshScripts` enumeration
0x01	Data	1	The `gxEnglishLanguage` constant of the `gxFontLanguage` enumeration
0x0001A	Data	26	The `length` field (short) of the `gxFontName` structure is 26 bytes.
0x41 70 70 6C 65 20 43 6F 6D 70 75 74 65 72 20 54 69 6D 65 73 20 52 6F 6D 61 6E	Data		Each of the 26 bytes is one glyph code. The font name is “Apple Computer Times Roman.”

Table 7-18 Analysis of the data stream of a flattened text shape (continued)

Values in data stream (binary)	Type of information	Value	Description
New style object			
0x01 (00 000001)	Operation opcode	0	New object
	Record size	1	Record size is 1 byte
0x28 (00 101000)	Compression type opcode	0	No compression
	Data type opcode	0x28	`gxStyleTypeOpcode` constant of the `gxGraphicsNewOpcode` enumeration
Set data for style object			
0x42 (01 000010)	Operation opcode	1	Set data.
	Record size	2	Record size is 2 bytes.
0x8A (10 001010)	Compression type opcode	2	Byte compression
	Data type opcode	10	`gxStyleFontOpcode` constant of the `gxStyleDataOpcode` enumeration
0x01	Data	1	A reference to font name object 1.
Set data for the text size of the style object			
0x43 (01 000011)	Operation opcode	1	Set data.
	Record size	3	Record size is 3 bytes.
0x49 (01 001001)	Compression type opcode	1	Word compression
	Data type opcode	9	`gxStyleTextSizeOpcode` constant of the `gxStyleDataOpcode` enumeration
0x00 87	Data	135.0	The size parameter for the `GXSetShapeTextSize` function is 135.0 points.
New ink object			
0x01 (00 000001)	Operation opcode	0	New object
	Record size	1	Record size is 1 byte.
0x29 (00 101001)	Compression type opcode	0	No compression
	Data type opcode	0x29	`gxInkTypeOpcode` constant of the `gxGraphicsNewOpcode` enumeration

continued

Table 7-18 Analysis of the data stream of a flattened text shape (continued)

Values in data stream (binary)	Type of information	Value	Description
Set data for ink color of the ink object			
0x47	Operation opcode	1	Set data.
(01 000111)	Record size	7	Record size is 7 bytes.
0x02	Compression type opcode	0	No compression
(00 000010)	Data type opcode	2	`gxInkColorOpcode` constant of the `gxInkDataOpcode` enumeration
0xB6 (10 11 0110)	Omit byte	–	The `gxOmitColorsMask` and `gxOmitColorsShift` enumerations are used to interpret this omit byte. Data1, color space, is byte compressed. Data2, color profile, is omitted so the default color profile is applied to the current object. Data3, color components, uses bits 3, 2, 1, and 0 for color space. The compression for each of the red, green and blue color components is byte compression.
0x03	Data1	3	`gxHSVSpace` constant of the `gxColorSpaces` enumeration
0x74 00	Data2	0.453	The hue of the `gxHSVColor` structure is 0.453.
0xFF	Data	0xFFFF	Since color components are 2-byte values, the byte is replicated to the value 0xFFFF. The saturation of the `gxHSVColor` structure is 1.0000.
0xFF	Data	0xFFFF	Since color components are 2-byte values, the byte is replicated to the value 0xFFFF. The value of the `gxHSVColor` structure is 1.0000.

New transform object

Bytes 0x01 and 0x2A define the new transform object. This data sequence is identical to the previous line shape example.

Table 7-18 Analysis of the data stream of a flattened text shape (continued)

Values in data stream (binary)	Type of information	Value	Description
Set data for mapping of the transform object			
0x59	Operation opcode	1	Set data.
(01 011001)	Record size	7	Record size is 25 bytes. The transform data size is 25 – 1 (data type opcode byte) = 24 bytes. Since each mapping requires 8 bytes, there are 24 / 8 = 3 mappings. This indicates that there is a translate, scale, and skew mapping.
0x03	Compression type opcode	0	No compression
(00 000011)	Data type opcode	3	`gxTransformMapping` constant of the `gxTransformDataOpcode` enumeration
0x003D0212	Data	61.12	The `deltaY` parameter for the `GXSetTransformMapping` function
0x000098FE	Data	0.60	The `deltaX` parameter for the `GXSetTransformMapping` function
0x0000F747	Data	0.97	The `hScale` parameter for the `GXSetTransformMapping` function
0x0000F747	Data	0.97	The `scale` parameter for the `GXSetTransformMapping` function
0x00004242	Data	0.26	The `hSkew` parameter for the `GXSetTransformMapping` function
0xFFFFBDBE	Data	–0.4242	The `vSkew` parameter for the `GXSetTransformMapping` function
New shape object			
0x09	Operation opcode	0	New object
(00 001001)	Record size	9	Record size is 9 bytes.
0x09	Compression type opcode	0	No compression
(00 001001)	Data type opcode	9	`gxTextType` constant of the `gxShapeTypes` enumeration

continued

Table 7-18 Analysis of the data stream of a flattened text shape (continued)

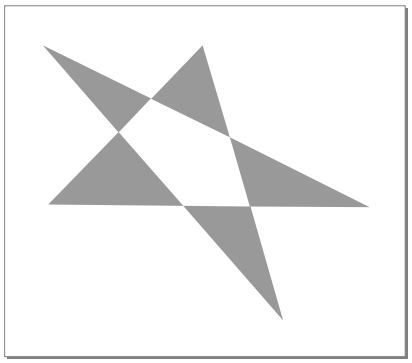
Values in data stream (binary)	Type of information	Value	Description
0x2A (00 10 10 10)	Omit byte	–	The gxOmitTextMask and gxOmitTextShift enumerations are used to interpret this omit byte. Byte compression is used for data1, and byte length. Byte compression is used for data2, and the x coordinate of the position. Word ?? compression is used for data3, y coordinate of position point. Byte compression is used for data4, number of characters and text.
0x02	Data1	2	The byte length is 2.
0x19	Data2	25.0000	The x-coordinate of the text position is 25.0000.
0x00 E6	Data3	230.0000	The y-coordinate of the text position is 230.0000.
0x02	Data4	2	The number of characters is 2.
0x47	Data4	0x47	Roman capital G
0x58	Data4	0x58	Roman capital X
Set data for attributes of the text object			
0x42	Operation opcode	2	Set data
(01 000010)	Record size	2	Record size is 2 bytes.
0x80	Compression type opcode	2	Byte compression
(10 000010)	Data type opcode	3	gxShapeAttributes constant of the gxShapeDataOpcode enumeration
0x20	Data	32	gxMapTransformShape constant of the gxShapeAttributes enumeration

Analyzing a Flattened Polygon Shape

The function described in the section “Creating a Picture With Seven Shapes” beginning on page 7-56 was first used to draw the picture shown in Figure 7-12 containing the polygon shape shown in Figure 7-18.

The yellow polygon shape is drawn with a pen thickness of 3.0 and skewed in the vertical direction by 0.5. Its size and shape is controlled by the vectors defined by the points (60.0, 0.0), (90.0, 90.0), (0.0, 30.0), (120.0, 30.0), (0.0, 90.0). The fill is even-odd. Once the polygon is drawn, it is moved to the point (240.0, 110.0) to position it in the picture.

Figure 7-18 The polygon shape drawn



The procedure described in the section “Flattening Shapes With GraphicsBug” beginning on page 7-54 was then used to generate the GraphicsBug output shown in Listing 7-9. The flattened polygon shape data stream is the sequential data that appears in parentheses.

Listing 7-9 GraphicsBug analysis of a flattened polygon shape

```
polygonType; byte compression (86)
(01 05 5a 01 2c 01 04 e2 97 5a 69 88 c4 78 00)
```

Table 7-19 shows the data stream analysis of the flattened polygon shape. The stream data is obtained from the GraphicsBug output in Listing 7-9. This table provides a description of each byte of the data stream for this shape. Data format sequences that are identical to previously described data sequences in the stream are not shown and are not analyzed here.

Table 7-19 Analysis of the data stream of a flattened polygon shape

Values in data stream (binary)	Type of information	Value	Description
New shape object			
`0x10` (`00 010000`)	Operation opcode	0	New object
	Record size	10	Record size is 10 bytes.
`0x86` (`10 000110`)	Compression type opcode	2	Byte compression
	Data type opcode	6	`gxPolygonType` constant of the `gxShapeTypes` enumeration
`0x01`	Data	1	The number of contours is 1.
`0x05`	Data	5	The number of vectors in the contour is 5.
`0x5A` (`01 01 10 10`)	Omit byte	–	The `gxOmitPathMask` and `gxOmitPathShift` enumerations are used to interpret this byte. Word compression is used for data1, and x coordinate of first point. Word compression is used for data2, and y coordinate of first point. Byte compression is used for data3, and all x relative coordinate deltas. Byte compression is used for data4, and all y relative coordinate deltas.
`0x01 2C`	Data1	290.0	Absolute x-coordinate of the first point is 290.0
`0x01 04`	Data2	260.0	Absolute y-coordinate of the first point is 260.0
`0xE2`	Data3	–30.0	The x-coordinate distance of the second point from the first point is –75.0. Absolute x coordinate of the second point is 290.0 – (–30.0) = 320.0
`0x97`	Data4	–105.0	The y-coordinate distance of the second point from the first point is –105.0. Absolute y-coordinate of the second point is 260.0 – (–105.0) = 365.0.
`0x5A`	Data3	90.0	The x-coordinate distance of the third point from the first point is 90.0. Absolute x-coordinate of the third point is 290.0 – (90.0) = 200.0.

Table 7-19 Analysis of the data stream of a flattened polygon shape (continued)

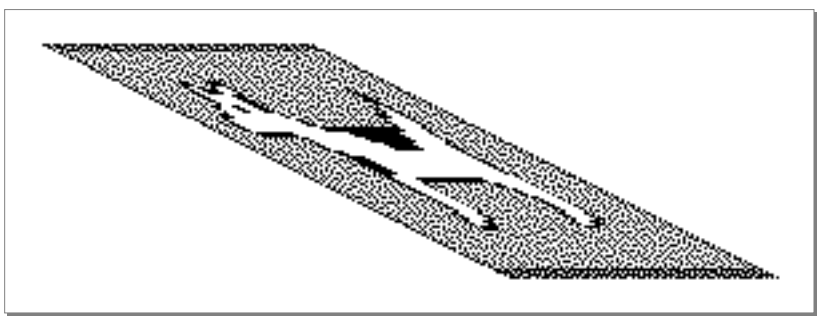
Values in data stream (binary)	Type of information	Value	Description
0x69	Data4	151.0	The y-coordinate distance of the third point from the first point is 151.0. Absolute y-coordinate of the third point is 260.0 – (151.0) = 109.0.
0x88	Data3	136.0	The x-coordinate distance of the fourth point from the first point is 70.0. Absolute x-coordinate of the fourth point is 290.0 – (70.0) = 220.0
0xC4	Data4	–60.0	The y-coordinate distance of the fourth point from the first point is –60.0. Absolute y-coordinate of the fourth point is 260.0 – (–60.0) = 320.0.
0x78	Data3	120.0	The x-coordinate distance of the fifth point from the first point is 70.0. Absolute x-coordinate of the fifth point is 290.0 – (120.0) = 170.0.
0x00	Data4	0.0	The y-coordinate distance of the fifth point from the first point is –50.0. Absolute y-coordinate of the fifth is 260.0 – (0.0) = 260.0.

Analyzing a Flattened Bitmap Shape

The function described in the section “Creating a Picture With Seven Shapes” beginning on page 7-56 was first used to draw the picture shown in Figure 7-12 containing the polygon shape shown in Figure 7-19.

The bitmap was retrieved from the resource fork and skewed in the horizontal direction by a factor of 2.0. Once the bitmap is drawn, it is moved to the point (200.0, 190.0) to position it in the picture.

Figure 7-19 The bitmap shape drawn



The procedure described in the section “Flattening Shapes With GraphicsBug” beginning on page 7-54 was then used to generate the GraphicsBug output shown in Listing 7-10. The flattened bitmap shape data stream is the sequential data that appears in parentheses.

Listing 7-10 GraphicsBug analysis of a flattened bitmap shape

```
newObject; size: #0 (01) [1]
transformType; no compression (2a)
setData; size: #12 (4d)
transformMapping; word compression (43)
(01 22 00 be 00 01 00 01 00 00 00 02)
newObject; size: #403 (00 00 01 94) [1]
bitImage; no compression (2e)
(a8 34 58 73 11 01 01 c2 81 70 22 01 21 82 ca ... )
newObject; size: #49 (32) [1]
colorSetType; byte compression (ac)
(01 ff ff ff ff 00 00 33 ff 00 33 cc 00 00 ...)
newObject; size: #10 (0b)
bitmapType; no compression (08)
(aa)
image (01)
width (66)
height (58)
rowBytes (34 ab)
pixelSize (04)
space (0b)
set (01 f0)
```

Table 7-20 shows the data stream analysis of the flattened bitmap shape. The stream data is obtained from the GraphicsBug output in Listing 7-10. This table provides a description of each byte of the data stream for this shape. Data format sequences that are identical to previously described data sequences in the stream are not shown and are not analyzed here.

Table 7-20 Analysis of the data stream of a bitmap shape

Values in data stream (binary)	Type of information	Value	Description
New transform object			
Bytes 0x01 and 0x2A define the new transform object. This data sequence is identical to the previous line shape example.			
Set data for mapping of the transform object			
0x4D (01 001101)	Operation opcode	1	Set data
	Record size	13	Record size is 13 bytes. The transform data size is13 – 1 (data type opcode byte) = 12 bytes. Since each mapping requires 8 bytes, there are 12 / 2 =6 mappings. This indicates that there is a translate, scale, and skew mapping.
0x43 (01 000011)	Compression type opcode	1	Word compression
	Data type opcode	3	`gxTransformMapping` constant of the `gxTransformDataOpcode` enumeration
0x0122	Data	290.0	The `deltaX` parameter for the `GXSetTransformMapping` function is 290.0.
0x00BE	Data	190.0	The `deltaY` parameter for the `GXSetTransformMapping` function is 190.0.
0x0001	Data	1.0	The `hScale` parameter for the `GXSetTransformMapping` function is 1.0.
0x0001	Data	1.0	The `vScale` parameter for the `GXSetTransformMapping` function is 1.0.
0x0000	Data	0.0	The `hSkew` parameter for the `GXSetTransformMapping` function is 0.0.
0x0002	Data	2.0	The `vSkew` parameter for the `GXSetTransformMapping` function is 2.0.

continued

Table 7-20 Analysis of the data stream of a bitmap shape (continued)

Values in data stream (binary)	Type of information	Value	Description
New bitmap image			
0x00 (00 000000)	Operation opcode	0	New object
	Record size	0	Record size is > 64 bytes.
0x00	Record size (continued)	0	Record size is > 256 bytes.
0x01 94	Record size (continued)	404	Record size is 404 bytes. For additional information about the stream format for the record size, see the section "Record Size" beginning on page 7-11.
0x2E (00 101110)	Compression type opcode	0	No compression
	Data type opcode	0x2E	`gxBitImageOpcode` constant of the `gxGraphicsNewOpcode` enumeration
0xA8 (10 10 1 000)	Omit byte	–	The `gxOmitBitImageMask` and `gxOmitBitImageShift` enumerations are used to interpret this omit byte. Data1, `width`, is byte compressed. Data2, `height`, is byte compressed. Data3, indicates that the bit image data is compressed. The last3 bits are not used and are reserved.
0x34	Data1	52	The bit image row width is 52 bytes.
0x58	Data2	88	The bit image column height is 88 bytes.
Row 1 of the bit image follows			
0x73 (01 110011)	Bit image compression byte	1	Bits 6 and 7 are 1. This is the `gxRepeatBitImageBytesOpcode` constant of the `gxBitImageCompression` enumeration.
		51	The bits that follow are to be repeated 51 times.
0x11	Data	11	The bits "11" are to be repeated 51 times

Table 7-20 Analysis of the data stream of a bitmap shape (continued)

Values in data stream (binary)	Type of information	Value	Description
0x01 (00 000001)	Data	0	This is the gxCopyBitImageBytesOpcode constant of the gxBitImageCompression enumeration. The bits in the next byte are added to the first row *x* number of times.
		1	The value of *x* is 1.
0x01	Data	"01"	The bits "01" are added to row 1
Rows 2 through 11 of the bit image follow			
0xC2 (11 000010)	Bit image compression byte	3	This is the gxRepeatBitImageScanOpcode constant of the gxBitImageCompression enumeration. The previous scan line is repeated *x* times.
	Previous row repeat number	2	The value of *x* is 2. The first row of bits is repeated 2 times.
Row 12			
0x81 (10 000001)	Bit image compression byte	2	This is the gxLookupBitImageBytesOpcode constant of the gxBitImageCompression enumeration. Repeat *x* bytes from the previous row and add them to the current row.
		1	The value of *x* is 1. One byte of data is to be repeated from the previous scan line.
0x70 01 110000	Bit image compression byte	1	Bits 6 and 7 are 1. This is the gxRepeatBitImageBytesOpcode constant of the gxBitImageCompression enumeration.
		48	The bits in the byte that follow are to be repeated 48 times.
0x22	Data	"100010"	The bits "100010" are to be repeated 48 times

continued

Table 7-20 Analysis of the data stream of a bitmap shape (continued)

Values in data stream (binary)	Type of information	Value	Description
0x01 (00 000001)	Bit image compression byte	0	This is the gxCopyBitImageBytesOpcode constant of the gxBitImageCompression enumeration. Repeat *x* bytes from the previous row and add them to the current row.
		1	The value of x is 1. One byte of data is to be repeated from the previous scan line.
0x21	Data	"100001"	The bits "100001" are to be repeated 1 time on the second row.
0x82 (10 000010)	Bit image compression byte	2	This is the gxLookupBitImageBytesOpcode constant of the gxBitImageCompression enumeration. Repeat x bytes from the previous row and add them to the current row.
		2	The value of *x* is 2. Two bytes of data is to be repeated from the previous scan line.
0xCA (11 001010)	Bit image compression byte	3	This is the gxRepeatBitImageScanOpcode constant of the gxBitImageCompression enumeration. The previous scan line is repeated *x* times.
		10	The value of *x* is 10. The first row of bits is repeated 10 times.
The remaining bytes of the bit image are not shown here.			
New color set object			
0x32 (00 110010)	Operation opcode	0	New object
	Record size	50	Record size is 50 bytes.
0xAC (10 101100)	Compression type opcode	2	Byte compression
	Data type opcode	3	gxColorSetTypeOpcode constant of the gxGraphicsNewOpcode enumeration
0x01	Data	1	gxRGBSpace constant of the gxColorSpaces enumeration

Table 7-20 Analysis of the data stream of a bitmap shape (continued)

Values in data stream (binary)	Type of information	Value	Description
White color for the bitmap object			
`0xFF`	Data	0xFFFF	Since color components are 2-byte values, the byte is replicated to the value 0xFFFF or 65,535. The RGB value for the `red` field of the `gxRgbColor` structure is 65,535.
`0xFF`	Data	0xFFFF	Since color components are 2-byte values, the byte is replicated to the value 0xFFFF or 65,535. The RGB value for the `green` field of the `gxRgbColor` structure is 65,535.
`0xFF`	Data	0xFFFF	Since color components are 2-byte values, the byte is replicated to the value 0xFFFF or 65,535. The RGB value for the `blue` field of the `gxRgbColor` structure is 65,535.
Dark blue color for the bitmap object			
`0x00`	Data	0x0000	Since color components are 2-byte values, the byte is replicated to the value 0x0000 or 0. The RGB value for the `red` field of the `gxRgbColor` structure is 0.
`0x00`	Data	0x0000	Since color components are 2-byte values, the byte is replicated to the value 0x0000 or 0. The RGB value for the `green` field of the `gxRgbColor` structure is 0.
`0x33`	Data	0x0000	Since color components are 2-byte values, the byte is replicated to the value 0x3333 or 0. The RGB value for the `blue` field of the `gxRgbColor` structure is 0x3333.
Cherry red color for the bitmap object			
`0xFF`	Data	0xFFFF	Since color components are 2-byte values, the byte is replicated to the value 0xFFFF or 65,535. The RGB value for the `red` field of the `gxRgbColor` structure is 65,535.

continued

Table 7-20 Analysis of the data stream of a bitmap shape (continued)

Values in data stream (binary)	Type of information	Value	Description
0x00	Data	0x0000	Since color components are 2-byte values, the byte is replicated to the value 0x0000 or 0. The RGB value for the `green` field of the `gxRgbColor` structure is 0.
0x33	Data	0x3333	Since color components are 2-byte values, the byte is replicated to the value 0x3333. The RGB value for the `blue` field of the `gxRgbColor` structure is 0x3333.
Dull red color for the bitmap object			
0xCC	Data	0xCCCC	Since color components are 2-byte values, the byte is replicated to the value 0xCCCC or 52,428. The RGB value for the `red` field of the `gxRgbColor` structure is 52,428.
0x00	Data	0x0000	Since color components are 2-byte values, the byte is replicated to the value 0x0000 or 0. The RGB value for the `green` field of the `gxRgbColor` structure is 0.
0x00	Data	0x0000	Since color components are 2-byte values, the byte is replicated to the value 0x0000 or 0. The RGB value for the `blue` field of the `gxRgbColor` structure is 0x0000.
The remaining 35 bytes of the color set are not shown here.			
New shape object			
0x10 (00 010000)	Operation opcode	0	New object
	Record size	11	Record size is 11 bytes.
0x08 (00 001000)	Compression type opcode	0	Byte compression
	Data type opcode	8	`gxBitmapType` constant of the `gxShapeTypes` enumeration
0xAA (10 10 10 10)	Omit byte	–	The `gxOmitBitmapMask1` and `gxOmitBitmapShift1` enumerations are used to interpret this byte. Byte compression is used for data1, data2, data3, and data4.
0x01	Data1	1	A pointer to the pixels located at 1.
0x66	Data2	102	The row width is 102 pixels.

Table 7-20 Analysis of the data stream of a bitmap shape (continued)

Values in data stream (binary)	Type of information	Value	Description
0x58	Data3	88	The column height is 88 pixels.
0x34	Data4	52	The row width is 52 bytes.
0xAB (10 10 10 11)	Omit byte	–	The `gxOmitBitmapMask2` and `gxOmitBitmapShift2` enumerations are used to interpret this byte. Byte compression is used for data1, data2, and data3. Data4 is omitted.
0x04	Data1	4	The number of bits per pixel is 1.
0x0B	Data2	11	`gxIndexedSpace` constant of the `gxColorSpaces` enumeration
0x01	Data3	1	The first set of bitmaps is used.
0xF0 (11 11 00 00)	Omit byte	–	The `gxOmitBitmapMask3` and `gxOmitBitmapShift3` enumerations are used to interpret this byte. Data1 and data2 are omitted. These are the x and y positions of the bitmap. The position is therefore at point (0, 0). The other bits are reserved.

Obtaining Data From a Print File

Any suitably equipped Macintosh computer with QuickDraw GX installed can read and print portable digital document print files created by your application. You may want to use the public data in a QuickDraw GX print file for other purposes. Listing 7-11 reads a portable digital document print file and returns the page count. For more information on print files and portable digital documents, see the chapters “Introduction to QuickDraw GX Printing” and “Core Printing Features” of *Inside Macintosh: QuickDraw GX Printing*.

Listing 7-11 Obtaining the page count from a portable digital document print file

```
#define nrequire( x, LABEL ) if((x)) goto LABEL

/* Returns the page count from an open print file */

   Parameters:-> short dataRefNum:reference to the spool file
                 <- long *pageCount:returns page count
   Returns:    OSErr
   Preconditions:dataRefNum != NULL
   Postconditions:none */
```

```
OSErr DespoolPageCount (short dataRefNum, long *pageCount);
OSErr DespoolPageCount (short dataRefNum, long *pageCount) {

   register OSErr anErr;
            long pageDirOffset, numPages;
            long dataLen;
/* position to read offset to page directory */

   anErr = SetFPos(dataRefNum, fsFromStart, (long) (kHeaderSize +
sizeof(long)));
   nrequire (anErr, SetPageDirOffsetPos);

   /* read offset to page directory */
   dataLen = sizeof(pageDirOffset);
   anErr = FSRead(dataRefNum, &dataLen, &pageDirOffset);
   nrequire (anErr, ReadPageDirOffsetPos);

/* move to page directory */
   anErr = SetFPos(dataRefNum, fsFromStart, (long) (pageDirOffset));
   nrequire (anErr, SetPageDirPos);

/* read number of pages */
   dataLen = sizeof(numPages);
   anErr = FSRead(dataRefNum, &dataLen, &numPages);
   nrequire (anErr, ReadNumPages);

   *pageCount = numPages;/* Return the result */

   ncheck (anErr);
   return anErr;

/* exception handling*/

ReadNumPages:
SetPageDirPos:
ReadPageDirOffsetPos:
SetPageDirOffsetPos:
   return anErr;
}
```

QuickDraw GX Stream Format Reference

This section provides reference information to the data structures and enumerations that are used in the stream format of a flattened shape.

Opcode Constants and Data Types

This section describes the constants and data types that describe the opcodes used in the data streams of flattened shapes.

Operation Opcode Byte

Bits 6 and 7 of the operation opcode byte are the operation opcode. This opcode provides a description of the data record that follows. Each operation opcode is defined in the `gxGraphicsOperationOpcode` enumeration.

```
enum gxGraphicsOperationOpcode {
   gxNewObjectOpcode = 0x00,
   gxSetDataOpcode   = 0x40,
   gxSetDefaultOpcode= 0x80,
   gxReservedOpcode  = 0xC0,
   gxNextOpcode      = 0xFF,
};
```

Constant descriptions

`gxNewObjectOpcode`
: Data for a new object follows.

`gxSetDataOpcode`
: Attributes for the current object follow.

`gxSetDefaultOpcode`
: Replace current default with the object that follows.

`gxReservedOpcode`
: This opcode is reserved for future expansion.

`gxNextOpcode`
: This constant is used by the current operand field to indicate that an opcode is coming.

Bits 0 through 5 of the operation opcode byte are the record size in bytes (1 to 63 bytes). The `gxObjectSizeMask` constant, binary 111111, masks bits 0 through 5 to select the record size. For additional information about the stream format for the record size, see the section “Record Size” beginning on page 7-11.

```
#define gxObjectSizeMask   0x3F
```

The `gxOpcodeShift` constant allows you to compare `gxGraphicsOperationOpcode` constants with other values.

```
#define gxOpcodeShift   6
```

Data Type Opcode Byte

Bits 6 and 7 of the data type opcode byte are the compression type opcode. The compression of the data to follow is given by the `gxTwoBitCompressionValues` enumeration in Table 7-3. The `gxCompressionMask` constant, binary 11, masks the constant defined by the `gxTwoBitCompressionValue` enumeration.

```
#define gxCompressionMask  0x03
```

The `gxCompressionShift` constant defines the number of bits to be shifted to the right so that the masked value of the compression type opcode can be compared to other values.

```
#define gxCompressionShift 6
```

Bits 0 through 5 of the data type opcode byte are the data type opcode. These opcodes describe the data that follows in the stream. The `gxObjectTypeMask` constant, binary 111111, masks bits 0 through 5 of the data type opcode byte to select the data type opcode. No shift is required to compare the data type opcode with other values.

```
#define gxObjectTypeMask   0x3F
```

Generic Data Opcode

The current operand uses a constant from the `gxGenericDataOpcode` enumeration when the current operand is the `gxNextOpcode` constant.

```
enum gxGenericDataOpcode {
   gxTypeOpcode,
   gxSizeOpcode
};
```

Constant descriptions

`gxTypeOpcode` The next opcode is a type opcode.

`gxSizeOpcode` The next opcode is a size opcode.

Bit Image Compression Opcode Byte

Bits 6 and 7 of the bit image compression opcode byte contain the compression type opcode that describes the data compression used for a region of a a bit image.The `gxBitimageOpcodeMask` constant, binary 11000000, masks bits 6 and 7 of the bit image compression opcode byte to select the bit image opcode.

```
#define gxBitimageOpcodeMask 0xC0
```

Once the `gxBitimageOpcodeMask` constant has been used to select the compression type opcode, a bit shift given by the `gxBitimageOpcodeShift` constant can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the compression type opcode so that it can be compared to other values.

```
#define gxBitimageOpcodeShift 6
```

Bits 0 through 5 of the bit image compression opcode byte contain the bit image count. This is the number of times that a binary sequence is repeated. The `gxBitimageCountMask` constant, binary 111111, masks bits 0 through 5 of the bit image compression opcode byte to select the bit image count. No shift is required to compare the bit image count with other values.

```
#define gxBitimageCountMask0x3F
```

Table 7-13 gives the bit image compression opcode constants. For additional information about the use of the bit image compression opcode byte, see the section “New Bit Image Object Data” beginning on page 7-49.

Modified Shape Data Opcodes

A constant from the `gxShapeDataOpcode` enumeration follows a `gxSetDataOpcode` operation opcode if shape data follows. The data stream bytes describe one of the fields specified in this enumeration.

```
enum gxShapeDataOpcode {
   gxShapeAttributesOpcode,
   gxShapeTagOpcode,
   gxShapeFillOpcode
};
```

Constant descriptions

`gxShapeAttributesOpcode`
: An attribute from the `gxShapeAttributes` enumeration is added to the current shape object.

`gxShapeTagOpcode`
: A tag is added to the current shape object.

`gxShapeFillOpcode`
: A fill is added to the current shape object.

Modified Style Data Opcodes

A constant from the `gxStyleDataOpcode` enumeration follows a `gxSetDataOpcode` if style data follows. The data stream bytes that follow describe one of the attributes specified in this enumeration.

```
enum gxStyleDataOpcode {
   gxStyleAttributesOpcode,
   gxStyleTagOpcode,
   gxStyleCurveErrorOpcode,
   gxStylePenOpcode,
   gxStyleJoinOpcode,
   gxStyleDashOpcode,
   gxStyleCapsOpcode,
   gxStylePatternOpcode,
   gxStyleTextAttributesOpcode,
   gxStyleTextSizeOpcode,
   gxStyleFontOpcode,
   gxStyleTextFaceOpcode,
   gxStylePlatformOpcode,
   gxStyleFontVariationsOpcode,
   gxStyleRunControlsOpcode,
   gxStyleRunPriorityJustOverrideOpcode,
   gxStyleRunGlyphJustOverridesOpcode,
   gxStyleRunGlyphSubstitutionsOpcode,
   gxStyleRunFeaturesOpcode,
   gxStyleRunKerningAdjustmentsOpcode,
   gxStyleJustificationOpcode
};
```

Constant descriptions

`gxStyleAttributesOpcode`
: The style attributes flags from the `gxStyleAttributes` enumeration follow.

`gxStyleTagOpcode`
: The parameters of the `GXSetStyleTags` function follow.

`gxStyleCurveErrorOpcode`
: Data for the `error` parameter of the `GXSetStyleCurveError` function follows.

`gxStylePenOpcode`
: The data for the `pen` parameter of the `GXSetStylePen` function follows.

`gxStyleJoinOpcode`
: The data for the fields of the `gxJoinRecord` structure follows.

`gxStyleDashOpcode`
The data for the fields of the `gxDashRecord` structure follows.

`gxStyleCapsOpcode`
The data for the fields of the `gxCapRecord` structure follows.

`gxStylePatternOpcode`
The data for the fields of the `gxPatternRecord` structure follows.

`gxStyleTextAttributesOpcode`
The data from the `gxTextAttributes` enumeration follows.

`gxStyleTextSizeOpcode`
The data for the `size` parameter of the `GXSetStyleTextSize` function follows.

`gxStyleFontOpcode`
The data for the `font` parameter of the `GXSetStyleFont` function follows.

`gxStyleTextFaceOpcode`
The data for the fields of the `gxTextFace` structure follows.

`gxStylePlatformOpcode`
The data for the parameters of the `GXStyleEncoding` function follows.

`gxStyleFontVariationsOpcode`
The data for the fields of the `gxFontVariations` structure follows.

`gxStyleRunControlsOpcode`
The data for the fields of the `gxRunControls` structure follows.

`gxStyleRunPriorityJustOverrideOpcode`
The data for the fields of the `gxPriorityJustificationOverride` structure follows.

`gxStyleRunGlyphJustOverridesOpcode`
The data for the fields of the `gxGlyphJustificationOverride` structure follows.

`gxStyleRunGlyphSubstitutionsOpcode`
The data for the fields of the `gxGlyphSubstitutionOverride` structure follows.

`gxStyleRunFeaturesOpcode`
The data for the fields of the `gxRunFeature` structure follows.

`gxStyleRunKerningAdjustmentsOpcode`
The data for the fields of the `gxKerningAdjustment` structure follows.

`gxStyleJustificationOpcode`
The data for the `justify` parameter of the `GXSetStyleJustification` function follows.

Modified Ink Data Opcodes

A constant from the `gxInkDataOpcode` enumeration follows a `gxSetDataOpcode` operation opcode if ink data follows. The data stream bytes that follow describe one of the attributes specified in this enumeration.

```
enum gxInkDataOpcode {
   gxInkAttributesOpcode,
   gxInkTagOpcode,
   gxInkColorOpcode,
   gxInkTransferModeOpcode
};
```

Constant descriptions

`gxInkAttributesOpcode`
: The parameters of the `GXSetInkAttributes` function follow.

`gxInkTagOpcode`
: The parameters of the `GXSetInkTags` function follow.

`gxInkColorOpcode`
: The parameters of the `GXSetInkColor` function follow.

`gxInkTransferModeOpcode`
: The parameters of the `GXSetInkTransfer` function follow.

Modified Color Set Data Opcodes

A constant from the `gxColorSetDataOpcode` enumeration follows a `gxSetDataOpcode` operation opcode if color set data follows. The bytes that follow describe one of the attributes specified in this enumeration.

```
enum gxColorSetDataOpcode {
   gxColorSetReservedOpcode,
   gxColorSetTagOpcode
};
```

Constant descriptions

`gxColorSetReservedOpcode`
: This opcode is reserved for future expansion.

`gxColorSetTagOpcode`
: The data parameters for the `GXSetColorSetTags` function follows.

Modified Color Profile Data Opcodes

A constant from the `gxProfileDataOpcode enumeration` follows a `gxSetDataOpcode` operation opcode if profile data follows. The data stream bytes that follow describe one of the attributes specified in this enumeration.

```
enum gxProfileDataOpcode {
   gxColorProfileAttributesOpcode,
   gxColorProfileTagOpcode
};
```

Constant descriptions

`gxColorProfileAttributesOpcode`
This opcode is reserved for future expansion.

`gxColorProfileTagOpcode`
The data parameters for the `GXSetColorProfileTags` function follow.

Modified Transform Data Opcodes

A constant from the `gxTransformDataOpcode` enumeration follows a `gxSetDataOpcode` operation opcode if transform data follows. The data stream bytes that follow describe one of the attributes specified in this enumeration.

```
enum  gxTransformDataOpcode{
   gxTransformReservedOpcode,
   gxTransformTagOpcode,
   gxTransformClipOpcode,
   gxTransformMappingOpcode,
   gxTransformPartMaskOpcode,
   gxTransformToleranceOpcode
};
```

Constant descriptions

`gxTransformReservedOpcode`
This opcode is reserved for future expansion.

`gxTransformTagOpcode`
The data parameters for the `GXSetTransformTags` function follow.

`gxTransformClipOpcode`
The data for the `clip` parameter of the `GXSetTransformClip` function follows.

`gxTransformMappingOpcode`
The data for the `map` parameter of the `GXSetTransformMapping` function follows.

`gxTransformPartMaskOpcode`
: The data for the `mask` parameter of the `GXSetTransformHitTest` function follows.

`gxTransformToleranceOpcode`
: The data for the `gxProfileRecord` structure and `gxProfileResponse` enumeration follows.

Bit Image Compression Opcodes

Bits 6 and 7 of the bit image compression opcode byte contain the bit image compression opcode. A constant from the `gxBitImageCompression` enumeration defines the compression of the bit image data sequence to immediately follow.

```
enum gxBitImageCompression {
   gxCopyBitImageBytesOpcode  = 0x00,
   gxRepeatBitImageBytesOpcode= 0x40,
   gxLookupBitImageBytesOpcode= 0x80,
   gxRepeatBitImageScanOpcode = 0xC0
};
```

Constant descriptions

`gxCopyBitImageBytesOpcode`
: Bit image compression opcode 0.

`gxRepeatBitImageBytesOpcode`
: Bit image compression opcode 1.

`gxLookupBitImageBytesOpcode`
: Bit image compression opcode 2.

`gxRepeatBitImageScanOpcode`
: Bit image compression opcode 3.

The bit image compression opcode is described in the section "New Bit Image Object Data" beginning on page 7-49.

Flatten Header Bytes

The two bytes following the byte containing the `gxHeaderTypeOpcode` contain the version of QuickDraw GX that generated the stream of data that follows and two flags that are defined by the `gxFlattenFlags` enumeration.

```
struct gxFlattenHeader {
   fixed          version;
   unsigned char  flatFlags;
};
```

Field descriptions

`version` The version of QuickDraw GX that was used to create the stream.

`flatFlags` The `gxFontListFlatten` and `gxFontGlyphsFlatten` flags.

The QuickDraw GX version and the flatten flags are described in the section "Header Data" beginning on page 7-27.

Style Object Omit Byte Constants and Data Types

This section describes the constants and data types that are used to interpret omit bytes that are used with style object data. The use of omit bytes is described in the section "Omit Byte Masks and Omit Byte Shifts" beginning on page 7-22.

Dash Style Omit Byte Masks and Shifts

The `gxOmitDashMask1` enumeration defines which bits in an omit byte correspond to the a data compression opcode for the field descriptors in the `gxDashRecord` structure. The sequence of data is also defined. The omit byte and its related data sequence are given in the section "Dash Data" beginning on page 7-37.

```
enum gxOmitDashMask1 {
   gxOmitDashAttributesMask   = 0xC0,
   gxOmitDashShapeMask        = 0x30,
   gxOmitDashAdvanceMask      = 0x0C,
   gxOmitDashPhaseMask        = 0x03
};
```

Constant descriptions

`gxOmitDashAttributesMask`
The mask to select the data compression bits for the `attributes` field descriptor.

`gxOmitDashShapeMask`
The mask to select the data compression bits for the `dash` field descriptor.

`gxOmitDashAdvanceMask`
The mask to select the data compression bits for the `advance` field descriptor.

`gxOmitDashPhaseMask`
The mask to select the data compression bits for the `phase` field descriptor.

Once one of the `gxOmitDashMask1` enumeration masks has been used to select data compression bits for a field descriptor in the `gxDashRecord` structure, the corresponding bit shift from the `gxOmitDashShift1` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression bits so that they can be compared to other values.

```
enum gxOmitDashShift1 {
   gxOmitDashAttributesShift  = 6,
   gxOmitDashShapeShift       = 4,
   gxOmitDashAdvanceShift     = 2,
   gxOmitDashPhaseShift       = 0
};
```

Constant descriptions

`gxOmitDashAttributesShift`
: The bit shift required to isolate the compression bits for the `attributes` field descriptor.

`gxOmitDashShapehift`
: The bit shift required to isolate the compression bits for the `dash` field descriptor.

`gxOmitDashAdvanceShift`
: The bit shift required to isolate the compression bits for the `advance` field descriptor.

`gxOmitDashPhaseShift`
: The bit shift required to isolate the compression bits for the `phase` field descriptor.

The `gxOmitDashMask2` enumeration defines which bits in a second omit byte correspond to the data compression bits for additional field descriptors in the `gxDashRecord` structure. The sequence of data is also continued. The use of this mask and shift are described in the section “Dash Data” beginning on page 7-37.

```
enum gxOmitDashMask2 {
   gxOmitDashScaleMask = 0xC0
};
```

Constant descriptions

`gxOmitDashScaleMask`
: The mask for the data compression bits for the `scale` field descriptor.

Once one of the `gxOmitDashMask2` enumeration masks has been used to select data compression bits for a field descriptor in the `gxDashRecord` structure, the corresponding bit shift from the `gxOmitDashShift2` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression bits so that they can be compared to other values.

```
enum gxOmitDashShift2{
      gxOmitDashScaleShift = 6
};
```

Constant descriptions

`gxOmitDashScaleShift`
The bit shift required to isolate the compression bits for the `scale` field descriptor.

Pattern Style Omit Byte Masks and Shifts

The `gxOmitPatternMask1` enumeration defines which bits in an omit byte correspond to the data compression opcodes for the field descriptors in the `gxPatternRecord` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Pattern Data" beginning on page 7-38.

```
enum gxOmitPatternMask1{

      gxOmitPatternAttributesMask   = 0xC0,
      gxOmitPatternShapeMask        = 0x30,
      gxOmitPatternUXMask           = 0x0C,
      gxOmitPatternUYMask           = 0x03
   };
```

Constant descriptions

`gxOmitPatternAttributesMask`
The mask used to select the data compression bits for the `attributes` field descriptor.

`gxOmitPatternShapeMask`
The mask used to select the data compression bits for the `pattern` field descriptor.

`gxOmitPatternUXMask`
The mask used to select the data compression bits for the `ux` field descriptor.

`gxOmitPatternUYMask`
The mask used to select the data compression bits for the `uy` field descriptor.

Once one of the `gxOmitPatternMask1` enumeration masks has been used to select data compression bits for one of the field descriptors in the `gxPatternRecord` structure, the corresponding bit shift from the `gxOmitPatternShift1` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression bits so that they can be compared to other values.

```
enum gxOmitPatternShift1 {
   gxOmitPatternAttributesShift  = 6,
   gxOmitPatternShapeShift       = 4,
   gxOmitPatternUXShift          = 2,
   gxOmitPatternUYShift          = 0
   };
```

Constant descriptions

`gxOmitPatternAttributesShift`
: The bit shift required to isolate the compression bits for the `attributes` field descriptor.

`gxOmitPatternShapeShift`
: The bit shift required to isolate the compression bits for the `pattern` field descriptor.

`gxOmitPatternUXShift`
: The bit shift required to isolate the compression bits for the `ux` field descriptor.

`gxOmitPatternUYShift`
: The bit shift required to isolate the compression bits for the `uy` field descriptor.

The `gxOmitPatternMask2` enumeration defines which bits in a second omit byte correspond to the data compression opcode for additional field descriptors in the `gxPatternRecord` structure. The sequence of data is also continued. The omit byte and its related data sequence is given in the section "Pattern Data" beginning on page 7-38.

```
enum gxOmitPatternMask2 {
   gxOmitPatternVXMask = 0xC0,
   gxOmitPatternVYMask = 0x30
};
```

Constant descriptions

`gxOmitPatternVXMask`
: The mask used to select the data compression bits for the u.x field descriptor.

`gxOmitPatternVYMask`
: The mask to select the data compression bits for the u.y field descriptor.

Once one of the `gxOmitPatternMask2` enumeration masks has been used to select a data compression opcode for one of the field descriptors in the `gxPatternRecord` structure, the corresponding bit shift from the `gxOmitPatternShift2` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitPatternShift2 {
   gxOmitPatternVXShift= 6,
   gxOmitPatternVYShift= 4
};
```

Constant descriptions

`gxOmitPatternVXShift`
: The bit shift required to isolate the compression bits for the u.x field descriptor.

`gxOmitPatternVYShift`
: The bit shift required to isolate the compression bits for the u.y field descriptor.

Join Style Omit Byte Masks and Shifts

The `gxOmitJoinMask` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxJoinRecord` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Join Data" beginning on page 7-37.

```
enum gxOmitJoinMask {
   gxOmitJoinAttributesMask= 0xC0,
   gxOmitJoinShapeMask     = 0x30,
   gxOmitJoinMiterMask     = 0x0C
};
```

Constant descriptions

`gxOmitJoinAttributesMask`
: The mask used to select the data compression bits for the `attributes` field descriptor.

`gxOmitJoinShapeMask`
: The mask used to select the data compression bits for the `join` field descriptor.

`gxOmitJoinMiterMask`
: The mask used to select the data compression bits for the `miter` field descriptor.

Once one of the `gxOmitJoinMask` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxJoinRecord` structure, the corresponding bit shift from the `gxOmitJoinShift` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitJoinShift {
   gxOmitJoinAttributesShift  = 6,
   gxOmitJoinShapeShift       = 4,
   gxOmitJoinMiterShift       = 2
};
```

Constant descriptions

`gxOmitJoinAttributesShift`
: The bit shift required to isolate the compression bits for the `attributes` field descriptor.

`gxOmitJoinShapeShift`
: The bit shift required to isolate the compression bits for the `join` field descriptor.

`gxOmitJoinMiterShift`
: The bit shift required to isolate the compression bits for the `miter` field descriptor.

Cap Style Omit Byte Masks and Shifts

The `gxOmitCapMask` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxCapRecord` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Caps Data" beginning on page 7-38.

```
enum gxOmitCapMask {
   gxOmitCapAttributesMask = 0xC0,
   gxOmitCapStartShapeMask = 0x30,
   gxOmitCapEndShapeMask   = 0x0C
};
```

Constant descriptions

`gxOmitCapAttributesMask`
: The mask used to select the data compression bits for the `attributes` field descriptor.

`gxOmitCapStartShapeMask`
: The mask used to select the data compression bits for the `startCap` field descriptor.

`gxOmitCapEndShapeMask`
: The mask used to select the data compression bits for the `endCap` field descriptor.

Once one of the `gxOmitCapMask` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxCapRecord` structure, the corresponding bit shift from the `gxOmitCapShift` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitCapShift {
   gxOmitCapAttributesShift= 6,
   gxOmitCapStartShapeShift= 4,
   gxOmitCapEndShapeShift  = 2
};
```

Constant descriptions

`gxOmitCapAttributesShift`
: The bit shift required to isolate the compression bits for the `attributes` field descriptor.

`gxOmitCapStartShapeShift`
: The bit shift required to isolate the compression bits for the `startCap` field descriptor.

`gxOmitCapEndShapeShift`
: The bit shift required to isolate the compression bits for the `endCap` field descriptor.

Text Face Style Omit Byte Masks and Shifts

The `gxOmitFaceMask` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxTextFace` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Text Face Data" beginning on page 7-39.

```
enum gxOmitFaceMask {
   gxOmitFaceLayersMask = 0xC0,
   gxOmitFaceMappingMask= 0x30
};
```

Constant descriptions

`gxOmitFaceLayersMask`
: The mask used to select the data compression bits for the `faceLayers` field descriptor.

`gxOmitFaceMappingMask`
: The mask used to select the data compression bits for the `advanceMapping` field descriptor.

Once one of the `gxOmitFaceMask` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxTextFace` structure, the corresponding bit shift from the `gxOmitFaceShift` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitFaceShift {
   gxOmitFaceLayersShift = 6,
   gxOmitFaceMappingShift= 4
};
```

Constant descriptions

`gxOmitFaceLayersShift`
: The bit shift required to isolate the compression bits for the `faceLayers` field descriptor.

`gxOmitFaceMappingShift`
: The bit shift required to isolate the compression bits for the `advanceMapping` field descriptor.

SEE ALSO

The section "Text Face Data" beginning on page 7-39 provides a full descrtiption of the `gxTextFace` structure.

Face Layer Omit Byte Masks and Shifts

The `gxOmitFaceLayerMask1` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxFaceLayer` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Text Face Data" on page 7-39.

```
enum gxOmitFaceLayerMask1 {
   gxOmitFaceLayerFillMask       = 0xC0,
   gxOmitFaceLayerFlagsMask      = 0x30,
   gxOmitFaceLayerStyleMask      = 0x0C,
   gxOmitFaceLayerTransformMask  = 0x03
};
```

Constant descriptions

`gxOmitFaceLayerFillMask`
: The mask used to select the data compression bits for the `outlineFill` field descriptor.

`gxOmitFaceLayerFlagsMask`
: The mask used to select the data compression bits for the `flags` field descriptor.

`gxOmitFaceLayerStyleMask`
: The mask used to select the data compression bits for the `outlineStyle` field descriptor.

`gxOmitFaceLayerTransformMask`
: The mask used to select the data compression bits for the `outlineTransform` field descriptor.

Once one of the `gxOmitFaceLayerMask1` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxFaceLayer` structure, the corresponding bit shift from the `gxOmitFaceLayerShift1` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitFaceLayerShift1 {
   gxOmitFaceLayerFillShift      = 6,
   gxOmitFaceLayerFlagsShift     = 4,
   gxOmitFaceLayerStyleShift     = 2,
   gxOmitFaceLayerTransformShift = 0
};
```

Constant descriptions

`gxOmitFaceLayerFillShift`
: The bit shift required to isolate the compression bits for the `outlineFill` field descriptor.

`gxOmitFaceLayerFlagsShift`
: The bit shift required to isolate the compression bits for the `flags` field descriptor.

`gxOmitFaceLayerStyleShift`
: The bit shift required to isolate the compression bits for the `outlineStyle` field descriptor.

`gxOmitFaceLayerTransformShift`
: The bit shift required to isolate the compression bits for the `outlineTransform` field descriptor.

The `gxOmitFaceLayerMask2` enumeration defines which bits in a second omit byte correspond to the data compression bits for additional field descriptors in the `gxFaceLayer` structure. The sequence of data is also defined. The use of this mask and shift are described in the section “Text Face Data” on page 7-39.

```
enum gxOmitFaceLayerMask2 {
   gxOmitFaceLayerBoldXMask   = 0xC0,
   gxOmitFaceLayerBoldYMask   = 0x30
};
```

Constant descriptions

`gxOmitFaceLayerBoldXMask`
: The mask used to select the data compression bits for the `boldOutset .X` field descriptor.

`gxOmitFaceLayerBoldYMask`
: The mask used to select the data compression bits for the `boldOutset .Y` field descriptor.

Once one of the `gxOmitFaceLayerMask2` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxFaceLayer` structure, the corresponding bit shift from the `gxOmitFaceLayerShift2` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitFaceLayerShift2 {
   gxOmitFaceLayerBoldXShift  = 6,
   gxOmitFaceLayerBoldYShift  = 4
};
```

Constant descriptions

`gxOmitFaceLayerBoldXShift`
: The bit shift required to isolate the compression bits for the `boldOutset .X` field descriptor.

`gxOmitFaceLayerBoldYShift`
: The bit shift required to isolate the compression bits for the `boldOutset .Y` field descriptor.

Ink Object Omit Byte Constants and Data Types

This section describes the constants and data types that are used to interpret omit bytes that are used with ink object data. The use of omit bytes is described in the section "Omit Byte Masks and Omit Byte Shifts" beginning on page 7-22.

Colors Omit Byte Masks and Shifts

The `gxOmitColorsMask` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxColor` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Color Data" beginning on page 7-44.

```
enum gxOmitColorsMask {
   gxOmitColorsSpaceMask      = 0xC0,
   gxOmitColorsProfileMask    = 0x30,
   gxOmitColorsComponentsMask = 0x0F,
   gxOmitColorsIndexMask      = 0x0C,
   gxOmitColorsIndexSetMask   = 0x03
};
```

Constant descriptions

`gxOmitColorsSpaceMask`
: The mask used to select the data compression bits for the `space` field descriptor.

`gxOmitColorsProfileMask`
: The mask used to select the data compression bits for the `profile` field descriptor.

`gxOmitColorsComponentsMask`
: The mask used to select the data compression bits for the `element.component[4]` field descriptor.

`gxOmitColorsIndexMask`
: The mask used to select the data compression bits for the `element.indexed.index` field descriptor.

`gxOmitColorsIndexSetMask`
: The mask used to select the data compression bits for the `element.index.Set` field descriptor.

Once one of the `gxOmitColorsMask` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxColor` structure, the corresponding bit shift from the `gxOmitColorsShift` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitColorsShift {
   gxOmitColorsSpaceShift        = 6,
   gxOmitColorsProfileShift      = 4,
   gxOmitColorsComponentsShift   = 0,
   gxOmitColorsIndexShift        = 2,
   gxOmitColorsIndexSetShift     = 0
};
```

Constant descriptions

`gxOmitColorsSpaceShift`
: The bit shift required to isolate the compression bits for the `space` field descriptor.

`gxOmitColorsProfileShift`
: The bit shift required to isolate the compression bits for the `profile` field descriptor.

`gxOmitColorsComponentsShift`
: The bit shift required to isolate the compression bits for the `element.component`[4] field descriptor.

`gxOmitColorsIndexShift`
: The bit shift required to isolate the compression bits for the `element.indexed.index` field descriptor.

`gxOmitColorsIndexSetShift`
: The bit shift required to isolate the compression bits for the `element.indexed.set` field descriptor.

Transfer Omit Byte Masks and Shifts

The `gxOmitTransferMask1` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxTransferMode` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Transfer Mode Data" beginning on page 7-44.

```
enum gxOmitTransferMask1 {
   gxOmitTransferSpaceMask    = 0xC0,
   gxOmitTransferSetMask      = 0x30,
   gxOmitTransferProfileMask  = 0x0C
};
```

Constant descriptions

`gxOmitTransferSpaceMask`
: The mask used to select the data compression bits for the `space` field descriptor.

`gxOmitTransferSetMask`
: The mask used to select the data compression bits for the `set` field descriptor.

`gxOmitTransferProfileMask`
: The mask used to select the data compression bits for the `profile` field descriptor.

Once one of the `gxOmitTransferMask1` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxTransferMode` structure, the corresponding bit shift from the `gxOmitTransferShift1` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitTransferShift1 {
   gxOmitTransferSpaceShift   = 6,
   gxOmitTransferSetShift     = 4,
   gxOmitTransferProfileShift = 2
};
```

Constant descriptions

`gxOmitTransferSpaceShift`
: The bit shift required to isolate the compression bits for the `space` field descriptor.

`gxOmitTransferSetShift`
: The bit shift required to isolate the compression bits for the `set` field descriptor.

`gxOmitTransferProfileShift`
: The bit shift required to isolate the compression bits for the `profile` field descriptor.

The `gxOmitTransferMask2` enumeration defines which bits in a second omit byte correspond to the data compression opcode for additional field descriptors in the `gxTransferMode` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Transfer Mode Data" beginning on page 7-44.

```
enum gxOmitTransferMask2 {
   gxOmitTransferSourceMatrixMask= 0xC0,
   gxOmitTransferDeviceMatrixMask= 0x30,
   gxOmitTransferResultMatrixMask= 0x0C,
   gxOmitTransferFlagsMask       = 0x03
};
```

Constant descriptions

`gxOmitTransferSourceMatrixMask`
: The mask used to select the data compression bits for the `sourceMatrix` field descriptor.

`gxOmitTransferDeviceMatrixMask`
: The mask used to select the data compression bits for the `deviceMatrix` field descriptor.

`gxOmitTransferResultMatrixMask`
: The mask used to select the data compression bits for the `resultMatrix` field descriptor.

`gxOmitTransferFlagsMask`
: The mask used to select the data compression bits for the `flags` field descriptor.

Once one of the `gxOmitTransferMask2` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxTransferMode` structure, the corresponding bit shift from the `gxOmitTransferShift2` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitTransferShift2 {

   gxOmitTransferSourceMatrixShift  = 6,
   gxOmitTransferDeviceMatrixShift  = 4,
   gxOmitTransferResultMatrixShift  = 2,
   gxOmitTransferFlagsShift         = 0
};
```

Constant descriptions

`gxOmitTransferSourceMatrixShift`
: The bit shift required to isolate the compression bits for the `sourceMatrix` field descriptor.

`gxOmitTransferDeviceMatrixShift`
: The bit shift required to isolate the compression bits for the `deviceMatrix` field descriptor.

`gxOmitTransferResultMatrixShift`
: The bit shift required to isolate the compression bits for the `resultMatrix` field descriptor.

`gxOmitTransferFlagsShift`
: The bit shift required to isolate the compression bits for the `flags` field descriptor.

Transfer Component Omit Byte Masks and Shifts

The `gxOmitTransferComponentMask1` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxTransferComponent` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Transfer Mode Data" beginning on page 7-44.

```
enum gxOmitTransferComponentMask1 {
   gxOmitTransferComponentModeMask           = 0x80,
   gxOmitTransferComponentFlagsMask          = 0x40,
   gxOmitTransferComponentSourceMinimumMask  = 0x30,
   gxOmitTransferComponentSourceMaximumMask  = 0x0C,
   gxOmitTransferComponentDeviceMinimumMask  = 0x03
};
```

Constant descriptions

`gxOmitTransferComponentModeMask`
: The mask used to select the data compression bits for the `mode` field descriptor.

`gxOmitTransferComponentFlagsMask`
: The mask used to select the data compression bits for the `flags` field descriptor.

`gxOmitTransferComponentSourceMinimumMask`
: The mask used to select the data compression bits for the `sourceMinimum` field descriptor.

`gxOmitTransferComponentSourceMaximumMask`
: The mask used to select the data compression bits for the `sourceMaximum` field descriptor.

`gxOmitTransferComponentDeviceMinimumMask`
: The mask used to select the data compression bits for the `deviceMinimum` field descriptor.

Once one of the `gxOmitTransferComponentMask1` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxTransferComponent` structure, the corresponding bit shift from the `gxOmitTransferComponentShift1` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitTransferComponentShift1 {
   gxOmitTransferComponentModeShift          = 7,
   gxOmitTransferComponentFlagsShift         = 6,
   gxOmitTransferComponentSourceMinimumShift = 4,
   gxOmitTransferComponentSourceMaximumShift = 2,
   gxOmitTransferComponentDeviceMinimumShift = 0
};
```

Constant descriptions

`gxOmitTransferComponentModeShift`
: The bit shift required to isolate the compression bits for the `mode` field descriptor.

`gxOmitTransferComponentFlagsShift`
: The bit shift required to isolate the compression bits for the `flags` field descriptor.

`gxOmitTransferComponentSourceMinimumShift`
: The bit shift required to isolate the compression bits for the `sourceMinimum` field descriptor.

`gxOmitTransferComponentSourceMaximumShift`
: The bit shift required to isolate the compression bits for the `sourceMaximum` field descriptor.

`gxOmitTransferComponentDeviceMinimumShift`
: The bit shift required to isolate the compression bits for the `deviceMinimum` field descriptor.

The `gxOmitTransferComponentMask2` enumeration defines which bits in a second omit byte correspond to the data compression opcode for additional field descriptors in the `gxTransferComponent` structure. The sequence of data is also continued. The omit byte and its related data sequence is given in the section “Transfer Mode Data” beginning on page 7-44.

```
enum gxOmitTransferComponentMask2 {
   gxOmitTransferComponentDeviceMaximumMask  = 0xC0,
   gxOmitTransferComponentClampMinimumMask   = 0x30,
   gxOmitTransferComponentClampMaximumMask   = 0x0C,
   gxOmitTransferComponentOperandMask        = 0x03
};
```

Constant descriptions

`gxOmitTransferComponentDeviceMaximumMask`
: The mask used to select the data compression bits for the `deviceMaximum` field descriptor.

`gxOmitTransferComponentClampMinimumMask`
: The mask used to select the data compression bits for the `clampMinimum` field descriptor.

`gxOmitTransferComponentClampMaximumMask`
: The mask used to select the data compression bits for the `clampMaximum` field descriptor.

`gxOmitTransferComponentOperandMask`
: The mask used to select the data compression bits for the `operand` field descriptor.

Once one of the `gxOmitTransferComponentMask2` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxTransferComponent` structure, the corresponding bit shift from the `gxOmitTransferComponentShift2` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitTransferComponentShift2 {
   gxOmitTransferComponentDeviceMaximumShift = 6,
   gxOmitTransferComponentClampMinimumShift  = 4,
   gxOmitTransferComponentClampMaximumShift  = 2,
   gxOmitTransferComponentOperandShift       = 0
};
```

Constant descriptions

`gxOmitTransferComponentDeviceMaximumShift`
: The bit shift required to isolate the compression bits for the `deviceMaximum` field descriptor.

`gxOmitTransferComponentClampMinimumShift`
: The bit shift required to isolate the compression bits for the `clampMinimum` field descriptor.

`gxOmitTransferComponentClampMaximumShift`
: The bit shift required to isolate the compression bits for the `clampMaximum` field descriptor.

`gxOmitTransferComponentOperandShift`
: The bit shift required to isolate the compression bits for the `operand` field descriptor.

Shape Object Omit Byte Constants and Data Types

This section describes the constants and data types that are used to interpret omit bytes that are used with shape object data. The use of omit bytes is described in the section "Omit Byte Masks and Omit Byte Shifts" beginning on page 7-22.

Path Shape Omit Byte Masks and Shifts

The `gxOmitPathMask` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxPaths` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Path Shape Data" beginning on page 7-31.

```
enum gxOmitPathMask {
   gxOmitPathPositionXMask = 0xC0,
   gxOmitPathPositionYMask = 0x30,
   gxOmitPathDeltaXMask    = 0x0C,
   gxOmitPathDeltaYMask    = 0x03
};
```

Constant descriptions

`gxOmitPathPositionXMask`
: The mask used to select the data compression bits for the `vector[0].x` field descriptor.

`gxOmitPathPositionYMask`
: The mask used to select the data compression bits for the `vector[0].y` field descriptor.

`gxOmitPathDeltaXMask`
: The mask used to select the data compression bits for the `vector[`*n*`].x` field descriptor where *n* is greater than zero, represented as a delta from the previous value.

`gxOmitPathDeltaYMask`
: The mask used to select the data compression bits for the `vector[`*n*`].y` field descriptor where *n* is greater than zero, represented as a delta from the previous value.

Once one of the `gxOmitPathMask` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxPaths??` structure, the corresponding bit shift from the `gxOmitPathShift` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitPathShift {
   gxOmitPathPositionXShift   = 6,
   gxOmitPathPositionYShift   = 4,
   gxOmitPathDeltaXShift      = 2,
   gxOmitPathDeltaYShift      = 0
};
```

Constant descriptions

`gxOmitPathPositionXShift`
: The bit shift required to isolate the compression bits for the `vector[0].x` field descriptor.

`gxOmitPathPositionYShift`
: The bit shift required to isolate the compression bits for the `vector[0].y` field descriptor.

`gxOmitPathDeltaXShift`
: The bit shift required to isolate the compression bits for the `vector[`*n*`].x` field descriptor where *n* is greater than zero, represented as a delta from the previous value.

`gxOmitPathDeltaYShift`
: The bit shift required to isolate the compression bits for the `vector[`*n*`].y` field descriptor where *n* is greater than zero, represented as a delta from the previous value.

Bitmap Shape Omit Byte Masks and Shifts

The `gxOmitBitmapMask1` enumeration defines which bits in an omit byte correspond to the data compression opcode for the field descriptors in the `gxBitmap` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Bitmap Shape Data" beginning on page 7-32.

```
enum gxOmitBitmapMask1 {
   gxOmitBitmapImageMask      = 0xC0,
   gxOmitBitmapWidthMask      = 0x30,
   gxOmitBitmapHeightMask     = 0x0C,
   gxOmitBitmapRowBytesMask   = 0x03
};
```

Constant descriptions

`gxOmitBitmapImageMask`
: The mask used to select the data compression bits for the `image` field descriptor.

`gxOmitBitmapWidthMask`
: The mask used to select the data compression bits for the `width` field descriptor.

`gxOmitBitmapHeightMask`
: The mask used to select the data compression bits for the `height` field descriptor.

`gxOmitBitmapRowBytesMask`
: The mask used to select the data compression bits for the `rowBytes` field descriptor.

Once one of the `gxOmitBitmapMask1` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxBitmap` structure, the corresponding bit shift from the `gxOmitBitmapShift1` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitBitmapShift1 {
   gxOmitBitmapImageShift     = 6,
   gxOmitBitmapWidthShift     = 4,
   gxOmitBitmapHeightShift    = 2,
   gxOmitBitmapRowBytesShift  = 0
};
```

Constant descriptions

`gxOmitBitmapImageShift`
: The bit shift required to isolate the compression bits for the `image` field descriptor.

`gxOmitBitmapWidthShift`
: The bit shift required to isolate the compression bits for the `width` field descriptor.

`gxOmitBitmapHeightShift`
: The bit shift required to isolate the compression bits for the `height` field descriptor.

`gxOmitBitmapRowBytesShift`
: The bit shift required to isolate the compression bits for the `rowBytes` field descriptor.

The `gxOmitBitmapMask2` enumeration defines which bits in a second omit byte correspond to the data compression opcode for additional field descriptors in the `gxBitmap` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section “Bitmap Shape Data” beginning on page 7-32.

```
enum gxOmitBitmapMask2 {
   gxOmitBitmapPixelSizeMask  = 0xC0,
   gxOmitBitmapSpaceMask      = 0x30,
   gxOmitBitmapSetMask        = 0x0C,
   gxOmitBitmapProfileMask    = 0x03
};
```

Constant descriptions

`gxOmitBitmapPixelSizeMask`
: The mask used to select the data compression bits for the `pixelSize` field descriptor.

`gxOmitBitmapSpaceMask`
: The mask used to select the data compression bits for the `space` field descriptor.

`gxOmitBitmapSetMask`
: The mask used to select the data compression bits for the `set` field descriptor.

`gxOmitBitmapProfileMask`
: The mask used to select the data compression bits for the `profile` field descriptor.

Once one of the `gxOmitBitmapMask2` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxBitmap` structure, the corresponding bit shift from the `gxOmitBitmapShift2` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitBitmapShift2 {
   gxOmitBitmapPixelSizeShift = 6,
   gxOmitBitmapSpaceShift     = 4,
   gxOmitBitmapSetShift       = 2,
   gxOmitBitmapProfileShift   = 0
};
```

Constant descriptions

`gxOmitBitmapPixelSizeShift`
: The bit shift required to isolate the compression bits for the `pixelSize` field descriptor.

`gxOmitBitmapSpaceShift`
: The bit shift required to isolate the compression bits for the `space` field descriptor.

`gxOmitBitmapSetShift`
: The bit shift required to isolate the compression bits for the `set` field descriptor.

`gxOmitBitmapProfileShift`
: The bit shift required to isolate the compression bits for the `profile` field descriptor.

The `gxOmitBitmapMask3` enumeration defines which bits in a third omit byte correspond to the data compression opcode for additional field descriptors in the `gxBitmap` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Bitmap Shape Data" beginning on page 7-32.

```
enum gxOmitBitmapMask3 {
   gxOmitBitmapPositionXMask = 0xC0,
   gxOmitBitmapPositionYMask = 0x30
};
```

Constant descriptions

`gxOmitBitmapPositionXMask`
: The mask used to select the data compression bits for the `positionX` field descriptor.

`gxOmitBitmapPositionYMask`
: The mask used to select the data compression bits for the `positionY` field descriptor.

Once one of the `gxOmitBitmapMask3` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxBitmap` structure, the corresponding bit shift from the `gxOmitBitmapShift3` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitBitmapShift3 {
   gxOmitBitmapPositionXShift = 6,
   gxOmitBitmapPositionYShift = 4
};
```

Constant descriptions

`gxOmitBitmapPositionXShift`
: The bit shift required to isolate the compression bits for the `positionX` field descriptor.

`gxOmitBitmapPositionYShift`
: The bit shift required to isolate the compression bits for the `positionY` field descriptor.

Bit Image Omit Byte Masks and Shifts

The `gxOmitBitImageMask` enumeration defines which bits in an omit byte correspond to the data compression opcode for additional field descriptors. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "New Bit Image Object Data" on page 7-49.

```
enum gxOmitBitImageMask {
   gxOmitBitImageRowBytesMask = 0xC0,
   gxOmitBitImageHeightMask   = 0x30,
   gxOmitBitImageDataMask     = 0x08
};
```

Constant descriptions

`gxOmitBitImageRowBytesMask`
: The mask used to select the data compression bits for the `rowBytes` field descriptor.

`gxOmitBitImageHeightMask`
: The mask used to select the data compression bits for the height.

`gxOmitBitImageDataMask`
: The mask used to select the data compression bits for the image.

Once one of the `gxOmitBitImageMask` enumeration masks has been used to select a data compression opcode for a field descriptor in the `gxBitmap` structure, the corresponding bit shift from the `gxOmitBitImageShift` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitBitImageShift {
   gxOmitBitImageRowBytesShift   = 6,
   gxOmitBitImageHeightShift     = 4,
   gxOmitBitImageDataShift       = 3
};
```

Constant descriptions

`gxOmitBitImageRowBytesShift`
: The bit shift required to isolate the compression bits for the `rowBytes` field descriptor.

`gxOmitBitImageHeightShift`
: The bit shift required to isolate the compression bits for the height.

`gxOmitBitImageDataShift`
: The bit shift required to isolate the compression bits for the image.

Text Shape Omit Byte Masks and Shifts

The `gxOmitTextMask` enumeration defines which bits in an omit byte correspond to the data compression opcode for parameters of the `GXNewText` function. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Text Shape Data" beginning on page 7-32.

```
enum gxOmitTextMask {
   gxOmitTextCharactersMask   = 0xC0,
   gxOmitTextPositionXMask    = 0x30,
   gxOmitTextPositionYMask    = 0x0C,
   gxOmitTextDataMask         = 0x02
};
```

Constant descriptions

`gxOmitTextCharactersMask`
: The mask used to select the data compression bits for the `charCount` parameter.

`gxOmitTextPositionXMask`
: The mask used to select the data compression bits for the `position.X` parameter.

`gxOmitTextPositionYMask`
: The mask used to select the data compression bits for the `position`.Y parameter.

`gxOmitTextDataMask`
: The mask used to select the data compression bits for the `text` parameter.

Once one of the `gxOmitText`Mask enumeration masks has been used to select a data compression opcode for the parameters of the `GXNewText` function, the corresponding bit shift from the `gxOmitTextShift` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitTextShift {
   gxOmitTextCharactersShift  = 6,
   gxOmitTextPositionXShift   = 4,
   gxOmitTextPositionYShift   = 2,
   gxOmitTextDataShift        = 1
};
```

Constant descriptions

`gxOmitTextCharactersShift`
: The bit shift required to isolate the compression bits for the `charCount` field descriptor.

`gxOmitTextPositionXShift`
: The bit shift required to isolate the compression bits for the `position.X` field descriptor.

`gxOmitTextPositionYShift`
: The bit shift required to isolate the compression bits for the `position`.Y field descriptor.

`gxOmitTextDataShift`
: The bit shift required to isolate the compression bits for the `text` field descriptor.

Glyph Shape Omit Byte Masks and Shifts

The `gxOmitGlyphMask1` enumeration defines which bits in an omit byte correspond to the data compression opcode for additional field descriptors in the `gx NewGlyphs` structure. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Glyph Shape Data" beginning on page 7-33.

```
enum gxOmitGlyphMask1 {
   gxOmitGlyphCharactersMask  = 0xC0,
   gxOmitGlyphLengthMask      = 0x30,
   gxOmitGlyphRunNumberMask   = 0x0C,
   gxOmitGlyphOnePositionMask = 0x02,
   gxOmitGlyphDataMask        = 0x01
};
```

Constant descriptions

`gxOmitGlyphCharactersMask`
: The mask used to select the data compression bits for the `charCount` function parameter.

`gxOmitGlyphLengthMask`
: The mask used to select the data compression bits for the length in bytes of the data.

`gxOmitGlyphRunNumberMask`
: The mask used to select the data compression bits for the number of `styleRuns`.

`gxOmitGlyphOnePositionMask`
: The mask used to specify that the position can be represented with one point.

`gxOmitGlyphDataMask`
: The mask used to select the data compression bits for the `text` function parameter.

Once one of the gxOmitGlyphMask1 enumeration masks has been used to select a data compression opcode for the parameters to GXNewGlyphs function, the corresponding bit shift from the gxOmitGlyphShift1 enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitGlyphShift1 {
   gxOmitGlyphCharactersShift    = 6,
   gxOmitGlyphLengthShift        = 4,
   gxOmitGlyphRunNumberShift     = 2,
   gxOmitGlyphOnePositionShift   = 1,
   gxOmitGlyphDataShift          = 0
};
```

Constant descriptions

gxOmitGlyphCharactersShift
: The bit shift required to isolate the compression bits for the charCount function parameter.

gxOmitGlyphLengthShift
: The bit shift required to isolate the compression bits for the length in bytes of the data.

gxOmitGlyphRunNumberShift
: The bit shift required to isolate the compression bits for the number of styleRuns.

gxOmitGlyphOnePositionShift
: The bit shift required to specify that the position can be represented with 1 point.

gxOmitGlyphDataShift
: The bit shift required to isolate the compression bits for the text function parameter.

The gxOmitGlyphMask2 enumeration defines which bits in an omit byte correspond to the data compression opcode for the parameters of the GXNewGlyphs function. The sequence of data is also defined. The omit byte and its related data sequence is given in the section “Glyph Shape Data” beginning on page 7-33.

```
enum gxOmitGlyphMask2 {
   gxOmitGlyphPositionsMask   = 0xC0,
   gxOmitGlyphAdvancesMask    = 0x20,
   gxOmitGlyphTangentsMask    = 0x18,
   gxOmitGlyphRunsMask        = 0x04,
   gxOmitGlyphStylesMask      = 0x03
};
```

Constant descriptions

`gxOmitGlyphPositionsMask`
: The mask used to select the data compression bits for the `positions` function parameter.

`gxOmitGlyphAdvancesMask`
: The mask used to select the data compression bits for the `advance` function parameter.

`gxOmitGlyphTangentsMask`
: The mask used to select the data compression bits for the `tangents` function parameter.

`gxOmitGlyphRunsMask`
: The mask used to select the data compression bits for the `styleRuns` function parameter.

`gxOmitGlyphStylesMask`
: The mask used to select the data compression bits for the `glyphStyles` function parameter.

Once one of the `gxOmitGlyphMask2` enumeration masks has been used to select a data compression opcode for the parameters to the `GXNewGlyph` function, the corresponding bit shift from the `gxOmitGlyphShift2` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitGlyphShift2 {
   gxOmitGlyphPositionsShift  = 6,
   gxOmitGlyphAdvancesShift   = 5,
   gxOmitGlyphTangentsShift   = 3,
   gxOmitGlyphRunsShift       = 2,
   gxOmitGlyphStylesShift     = 0
};
```

Constant descriptions

`gxOmitGlyphPositionsShift`
: The bit shift required to isolate the compression bits for the `positions` function parameter.

`gxOmitGlyphAdvancesShift`
: The bit shift required to isolate the compression bits for the `advance` function parameter.

`gxOmitGlyphTangentsShift`
: The bit shift required to isolate the compression bits for the `tangents` function parameter.

`gxOmitGlyphRunsShift`
: The bit shift required to isolate the compression bits for the `styleRuns` function parameter.

`gxOmitGlyphStylesShift`
: The bit shift required to isolate the compression bits for the `glyphStyles` function parameter.

Layout Shape Omit Byte Masks and Shifts

The `gxOmitLayoutMask1` enumeration defines which bits in an omit byte correspond to the data compression opcode for parameters for the `GXNewLayout` function. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Layout Shape Data" beginning on page 7-33.

```
enum gxOmitLayoutMask1 {
   gxOmitLayoutLengthMask     = 0xC0,
   gxOmitLayoutPositionXMask  = 0x30,
   gxOmitLayoutPositionYMask  = 0x0C,
   gxOmitLayoutDataMask       = 0x02
};
```

Constant descriptions

`gxOmitLayoutLengthMask`
: The mask used to select the data compression bits for the `textRunLength` parameter.

`gxOmitLayoutPositionXMask`
: The mask used to select the data compression bits for the `position.X` parameter.

`gxOmitLayoutPositionYMask`
: The mask used to select the data compression bits for the `position.Y` parameters.

`gxOmitLayoutDataMask`
: The mask used to select the data compression bits for the `text` parameter.

Once one of the `gxOmitLayoutMask1` enumeration masks has been used to select a data compression opcode for the parameters for the `GXNewLayout` function, the corresponding bit shift from the `gxOmitLayoutShift1` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitLayoutShift1 {
   gxOmitLayoutLengthShift    = 6,
   gxOmitLayoutPositionXShift = 4,
   gxOmitLayoutPositionYShift = 2,
   gxOmitLayoutDataShift      = 1
};
```

Constant descriptions

`gxOmitLayoutLengthShift`
: The bit shift required to isolate the compression bits for the `textRunLength` parameter.

`gxOmitLayoutPositionXShift`
: The bit shift required to isolate the compression bits for the `position.X` parameter.

`gxOmitLayoutPositionYShift`
: The bit shift required to isolate the compression bits for the `position.Y` parameter.

`gxOmitLayoutDataShift`
: The bit shift required to isolate the compression bits for the `text` parameter descriptor.

The `gxOmitLayoutMask2` enumeration defines which bits in a second omit byte correspond to the data compression opcode for additional parameters for the `GXNewLayout` function. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Layout Shape Data" beginning on page 7-33.

```
enum gxOmitLayoutMask2 {
   gxOmitLayoutWidthMask   = 0xC0,
   gxOmitLayoutFlushMask   = 0x30,
   gxOmitLayoutJustMask    = 0x0C,
   gxOmitLayoutOptionsMask = 0x03
};
```

Constant descriptions

`gxOmitLayoutWidthMask`
: The mask used to select the data compression bits for the `width` field descriptor.

`gxOmitLayoutFlushMask`
: The mask used to select the data compression bits for the `flush` field descriptor.

`gxOmitLayoutJustMask`
: The mask used to select the data compression bits for the `just` field descriptor.

`gxOmitLayoutOptionsMask`
: The mask used to select the data compression bits for the `flags` field descriptor.

Once one of the gxOmitLayoutMask2 enumeration masks has been used to select a data compression opcode for the parameters for the GXNewLayout function, the corresponding bit shift from the gxOmitLayoutShift2 enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitLayoutShift2 {
   gxOmitLayoutWidthShift     = 6,
   gxOmitLayoutFlushShift     = 4,
   gxOmitLayoutJustShift      = 2,
   gxOmitLayoutOptionsShift   = 0
};
```

Constant descriptions

gxOmitLayoutWidthShift
: The bit shift required to isolate the compression bits for the width field descriptor.

gxOmitLayoutFlushShift
: The bit shift required to isolate the compression bits for the flush field descriptor.

gxOmitLayoutJustShift
: The bit shift required to isolate the compression bits for the just field descriptor.

gxOmitLayoutOptionsShift
: The bit shift required to isolate the compression bits for the flags field descriptor.

The gxOmitLayoutMask3 enumeration defines which bits in a third omit byte correspond to the data compression opcode for additional parameters for the GXNewLayout function. The sequence of data is also defined. The omit byte and its related data sequence is given in the section “Layout Shape Data” beginning on page 7-33.

```
enum gxOmitLayoutMask3 {
   gxOmitLayoutStyleRunNumberMask= 0xC0,
   gxOmitLayoutLevelRunNumberMask= 0x30,
   gxOmitLayoutHasBaselineMask   = 0x08,
   gxOmitLayoutStyleRunsMask     = 0x04,
   gxOmitLayoutStylesMask        = 0x03
};
```

Constant descriptions

`gxOmitLayoutStyleRunNumberMask`
: The mask used to select the data compression bits for the `styleRunCount` field descriptor.

`gxOmitLayoutLevelRunNumberMask`
: The mask used to select the data compression bits for the `levelRunCount` field descriptor.

`gxOmitLayoutHasBaselineMask`
: The mask used to select the data compression bits for the `hasBaseline` field descriptor.

`gxOmitLayoutStyleRunsMask`
: The mask used to select the data compression bits for the `styleRunLengths` field descriptor.

`gxOmitLayoutStylesMask`
: The mask used to select the data compression bits for the ??? field descriptor.

Once one of the `gxOmitLayoutMask3` enumeration masks has been used to select a data compression opcode for the parameters for the `GXNewLayout` function, the corresponding bit shift from the `gxOmitLayoutShift3` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitLayoutShift3 {
   gxOmitLayoutStyleRunNumberShift  = 6,
   gxOmitLayoutLevelRunNumberShift  = 4,
   gxOmitLayoutHasBaselineShift     = 3,
   gxOmitLayoutStyleRunsShift       = 2,
   gxOmitLayoutStylesShift          = 0
};
```

Constant descriptions

`gxOmitLayoutStyleRunNumberShift`
: The bit shift required to isolate the compression bits for the `styleRunCount` field descriptor.

`gxOmitLayoutLevelRunNumberShift`
: The bit shift required to isolate the compression bits for the `levelRunCount` field descriptor.

`gxOmitLayoutHasBaselineShift`
: The bit shift required to isolate the compression bits for the `hasBaseline` field descriptor.

`gxOmitLayoutStyleRunsShift`
: The bit shift required to isolate the compression bits for the `styleRunLengths` field descriptor.

`gxOmitLayoutStylesShift`
: The bit shift required to isolate the compression bits for the ??? field descriptor.

The `gxOmitLayoutMask4` enumeration defines which bits in a fourth omit byte correspond to the data compression opcode for additional parameters for the `GXNewLayout` function. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Layout Shape Data" beginning on page 7-33.

```
enum gxOmitLayoutMask4 {

   gxOmitLayoutLevelRunsMask     = 0x80,
   gxOmitLayoutLevelsMask        = 0x40
};
```

Constant descriptions

`gxOmitLayoutLevelRunsMask`
: The mask used to select the data compression bits for the `levelRunLengths` parameter.

`gxOmitLayoutLevelsMask`
: The mask used to select the data compression bits for the `levels` parameter.

Once one of the `gxOmitLayoutMask4` enumeration masks has been used to a select data compression opcode for the parameters for the `GXNewLayout` function, the corresponding bit shift from the `gxOmitLayoutShift4` enumeration can be applied to the selected bits. The selected bits must be moved to the right by the indicated number of bits to isolate the data compression opcode so that it can be compared to other values.

```
enum gxOmitLayoutShift4 {
   gxOmitLayoutLevelRunsShift = 7,
   gxOmitLayoutLevelsShift    = 6
};
```

Constant descriptions

`gxOmitLayoutLevelRunsShift`
: The bit shift required to isolate the compression bits for the `levelRunLengths` parameter.

`gxOmitLayoutLevelsShift`
: The bit shift required to isolate the compression bits for the `levels` parameter.

Picture Shape Omit Byte Masks and Shifts

The `gxOmitPictureParametersMask` enumeration defines which bits in an omit byte correspond to the data compression opcode for parameters of the `GXDrawPicture` function. The sequence of data is also defined. The omit byte and its related data sequence is given in the section "Picture Shape Data" beginning on page 7-34.

```
enum gxOmitPictureParametersMask {
   gxOmitPictureShapeMask        = 0xC0,
   gxOmitOverrideStyleMask       = 0x30,
   gxOmitOverrideInkMask         = 0x0C,
   gxOmitOverrideTransformMask   = 0x03
};
```

Constant descriptions

`gxOmitPictureShapeMask`
: The mask used to select the data compression bits for the `shapes` parameter.

`gxOmitOverrideStyleMask`
: The mask used to select the data compression bits for the `styles` parameter.

`gxOmitOverrideInkMask`
: The mask used to select the data compression bits for the `inks` parameter.

`gxOmitOverrideTransformMask`
: The mask used to select the data compression bits for the `transforms` parameter.

```
enum gxOmitPictureParametersShift {
   gxOmitPictureShapeShift       = 0x6,
   gxOmitOverrideStyleShift      = 0x4,
   gxOmitOverrideInkShift        = 0x2,
   gxOmitOverrideTransformShift  = 0x0
};
```

Constant descriptions

`gxOmitPictureShapeShift`
: The bit shift required to isolate the compression bits for the `shapes` parameter.

`gxOmitOverrideStyleShift`
: The bit shift required to isolate the compression bits for the `styles` parameter.

`gxOmitOverrideInkShift`
: The bit shift required to isolate the compression bits for the `inks` parameter.

`gxOmitOverrideTransformShift`
: The bit shift required to isolate the compression bits for the `transforms` parameter.

QuickDraw GX Stream Format Summary

Opcode Constants and Data Types

Operation Opcode Byte

```
enum gxGraphicsOperationOpcode {
   gxNewObjectOpcode = 0x00,
   gxSetDataOpcode   = 0x40,
   gxSetDefaultOpcode= 0x80,
   gxReservedOpcode  = 0xC0,
   gxNextOpcode      = 0xFF,
};
```

Data Type Opcode Byte

```
enum gxGraphicsNewOpcode {
   gxHeaderTypeOpcode      = 0x00,
   gxStyleTypeOpcode    = 0x28,
   gxInkTypeOpcode,
   gxTransformTypeOpcode,
   gxColorProfileTypeOpcode,
   gxColorSetTypeOpcode,
   gxTagTypeOpcode,
   gxBitImageOpcode,
   gxFontNameTypeOpcode,
   gxTrailerTypeOpcode,
};
```

Generic Data Opcode

```
enum gxGenericDataOpcode {
   gxTypeOpcode,
   gxSizeOpcode
};                      /* constants used by current operand when
                           current operation is gxNextOpcode */
#define gxCompressionShift       6
#define gxObjectTypeMask      0x3F
```

```
#define gxBitImageOpcodeMask  0xC0
#define gxBitImageCountMask   0x3F
#define gxBitImageOpcodeShift    6
```

Modified Shape Data Opcodes

```
enum gxShapeDataOpcode {
   gxShapeAttributesOpcode,
   gxShapeTagOpcode,
   gxShapeFillOpcode
};
```

Modified Style Data Opcodes

```
enum gxStyleDataOpcode {
   gxStyleAttributesOpcode,
   gxStyleTagOpcode,
   gxStyleCurveErrorOpcode,
   gxStylePenOpcode,
   gxStyleJoinOpcode,
   gxStyleDashOpcode,
   gxStyleCapsOpcode,
   gxStylePatternOpcode,
   gxStyleTextAttributesOpcode,
   gxStyleTextSizeOpcode,
   gxStyleFontOpcode,
   gxStyleTextFaceOpcode,
   gxStylePlatformOpcode,
   gxStyleFontVariationsOpcode,
   #ifdef gxLayoutStyleRuns
   gxStyleRunControlsOpcode,
   gxStyleRunPriorityJustOverrideOpcode,
   gxStyleRunGlyphJustOverridesOpcode,
   gxStyleRunGlyphSubstitutionsOpcode,
   gxStyleRunFeaturesOpcode,
   gxStyleRunKerningAdjustmentsOpcode,
   gxStyleLayoutInfoOpcode,
   gxStyleJustificationOpcode
};
```

Modified Ink Data Opcodes

```
enum gxInkDataOpcode {
   gxInkAttributesOpcode,
   gxInkTagOpcode,
   gxInkColorOpcode,
   gxInkTransferModeOpcode
};
```

Modified Color Set Data Opcodes

```
enum gxColorSetDataOpcode {
   gxColorSetReservedOpcode,
   gxColorSetTagOpcode
};
```

Modified Color Profile Data Opcodes

```
enum gxProfileDataOpcode {
   gxColorProfileAttributesOpcode,
   gxColorProfileTagOpcode
};
```

Modified Transform Data Opcodes

```
enum gxTransformDataOpcode {
   gxTransformReservedOpcode,
   gxTransformTagOpcode,
   gxTransformClipOpcode,
   gxTransformMappingOpcode,
   gxTransformPartMaskOpcode,
   gxTransformToleranceOpcode
};
```

Bit Image Compression Opcodes

```
enum gxBitImageCompression {
   gxCopyBitImageBytesOpcode  = 0x00,
   gxRepeatBitImageBytesOpcode= 0x40,
   gxLookupBitImageBytesOpcode= 0x80,
   gxRepeatBitImageScanOpcode = 0xC0
};
```

Two Bit Compression Values

```
enum gxTwoBitCompressionValues {
   gxNoCompression,     = 0x00
   gxWordCompression,   = 0x40
   gxByteCompression,   = 0x80
   gxOmitCompression =  = 0x??
};
```

Flatten Header Bytes

```
struct gxFlattenHeader {
   fixed          version;
   unsigned char  flatFlags;
};
```

Style Object Omit Byte Constants and Data Types

Dash Style Omit Byte Masks and Shifts

```
enum gxOmitDashMask1 {
   gxOmitDashAttributesMask   = 0xC0,
   gxOmitDashShapeMask        = 0x30,
   gxOmitDashAdvanceMask      = 0x0C,
   gxOmitDashPhaseMask        = 0x03
};

enum gxOmitDashShift1 {
   gxOmitDashAttributesShift  = 6,
   gxOmitDashShapeShift       = 4,
   gxOmitDashAdvanceShift     = 2,
   gxOmitDashPhaseShift       = 0
};

enum gxOmitDashMask2 {
   gxOmitDashScaleMask        =  0xC0
};

enum gxOmitDashShift2 {
      gxOmitDashScaleShift    = 6
   };
```

Pattern Style Omit Byte Masks and Shifts

```
enum gxOmitPatternMask1 {

      gxOmitPatternAttributesMask   = 0xC0,
      gxOmitPatternShapeMask        = 0x30,
      gxOmitPatternUXMask           = 0x0C,
      gxOmitPatternUYMask           = 0x03
   };

enum gxOmitPatternShift1 {
   gxOmitPatternAttributesShift  = 6,
   gxOmitPatternShapeShift       = 4,
   gxOmitPatternUXShift          = 2,
   gxOmitPatternUYShift          = 0
   };

enum gxOmitPatternMask2 {
   gxOmitPatternVXMask = 0xC0,
   gxOmitPatternVYMask = 0x30
};

enum gxOmitPatternShift2 {
   gxOmitPatternVXShift= 6,
   gxOmitPatternVYShift= 4
};
```

Join Style Omit Byte Masks and Shifts

```
enum gxOmitJoinMask {
   gxOmitJoinAttributesMask= 0xC0,
   gxOmitJoinShapeMask     = 0x30,
   gxOmitJoinMiterMask     = 0x0C
};

enum gxOmitJoinShift {
   gxOmitJoinAttributesShift  = 6,
   gxOmitJoinShapeShift       = 4,
   gxOmitJoinMiterShift       = 2
};
```

Cap Style Omit Byte Masks and Shifts

```
enum gxOmitCapMask {
   gxOmitCapAttributesMask = 0xC0,
   gxOmitCapStartShapeMask = 0x30,
   gxOmitCapEndShapeMask   = 0x0C
};

enum gxOmitCapShift {
   gxOmitCapAttributesShift= 6,
   gxOmitCapStartShapeShift= 4,
   gxOmitCapEndShapeShift  = 2
};
```

Text Face Style Omit Byte Masks and Shifts

```
enum gxOmitFaceMask {
   gxOmitFaceLayersMask = 0xC0,
   gxOmitFaceMappingMask= 0x30
};

enum gxOmitFaceShift {
   gxOmitFaceLayersShift = 6,
   gxOmitFaceMappingShift= 4
};
```

Face Layer Omit Byte Masks and Shifts

```
enum gxOmitFaceLayerMask1 {
   gxOmitFaceLayerFillMask       = 0xC0,
   gxOmitFaceLayerFlagsMask      = 0x30,
   gxOmitFaceLayerStyleMask      = 0x0C,
   gxOmitFaceLayerTransformMask  = 0x03
};

enum gxOmitFaceLayerShift1 {
   gxOmitFaceLayerFillShift      = 6,
   gxOmitFaceLayerFlagsShift     = 4,
   gxOmitFaceLayerStyleShift     = 2,
   gxOmitFaceLayerTransformShift = 0
};
```

```
enum gxOmitFaceLayerMask2 {
   gxOmitFaceLayerBoldXMask   = 0xC0,
   gxOmitFaceLayerBoldYMask   = 0x30
};

enum gxOmitFaceLayerShift2 {
   gxOmitFaceLayerBoldXShift  = 6,
   gxOmitFaceLayerBoldYShift  = 4
};
```

Ink Object Omit Byte Constants and Data Types

Colors Omit Byte Masks and Shifts

```
enum gxOmitColorsMask {

   gxOmitColorsSpaceMask      = 0xC0,
   gxOmitColorsProfileMask    = 0x30,
   gxOmitColorsComponentsMask = 0x0F,
   gxOmitColorsIndexMask      = 0x0C,
   gxOmitColorsIndexSetMask   = 0x03
};

enum gxOmitColorsShift {
   gxOmitColorsSpaceShift        = 6,
   gxOmitColorsProfileShift      = 4,
   gxOmitColorsComponentsShift   = 0,
   gxOmitColorsIndexShift        = 2,
   gxOmitColorsIndexSetShift     = 0
};
```

Transfer Omit Byte Masks and Shifts

```
enum gxOmitTransferMask1 {
   gxOmitTransferSpaceMask    = 0xC0,
   gxOmitTransferSetMask      = 0x30,
   gxOmitTransferProfileMask  = 0x0C
};

enum gxOmitTransferShift1 {
   gxOmitTransferSpaceShift   = 6,
   gxOmitTransferSetShift     = 4,
   gxOmitTransferProfileShift = 2
};
```

```
enum gxOmitTransferMask2 {
   gxOmitTransferSourceMatrixMask= 0xC0,
   gxOmitTransferDeviceMatrixMask= 0x30,
   gxOmitTransferResultMatrixMask= 0x0C,
   gxOmitTransferFlagsMask       = 0x03
};

enum gxOmitTransferShift2 {

   gxOmitTransferSourceMatrixShift  = 6,
   gxOmitTransferDeviceMatrixShift  = 4,
   gxOmitTransferResultMatrixShift  = 2,
   gxOmitTransferFlagsShift         = 0
};
```

Transfer Component Omit Byte Masks and Shifts

```
enum gxOmitTransferComponentMask1{
   gxOmitTransferComponentModeMask           = 0x80,
   gxOmitTransferComponentFlagsMask          = 0x40,
   gxOmitTransferComponentSourceMinimumMask  = 0x30,
   gxOmitTransferComponentSourceMaximumMask  = 0x0C,
   gxOmitTransferComponentDeviceMinimumMask  = 0x03
} ;

enum gxOmitTransferComponentShift1 {
   gxOmitTransferComponentModeShift          = 7,
   gxOmitTransferComponentFlagsShift         = 6,
   gxOmitTransferComponentSourceMinimumShift = 4,
   gxOmitTransferComponentSourceMaximumShift = 2,
   gxOmitTransferComponentDeviceMinimumShift = 0
};

enum gxOmitTransferComponentMask2 {
   gxOmitTransferComponentDeviceMaximumMask  = 0xC0,
   gxOmitTransferComponentClampMinimumMask   = 0x30,
   gxOmitTransferComponentClampMaximumMask   = 0x0C,
   gxOmitTransferComponentOperandMask        = 0x03
};
```

```
enum gxOmitTransferComponentShift2 {
   gxOmitTransferComponentDeviceMaximumShift = 6,
   gxOmitTransferComponentClampMinimumShift  = 4,
   gxOmitTransferComponentClampMaximumShift  = 2,
   gxOmitTransferComponentOperandShift       = 0
};
```

Shape Object Omit Byte Constants and Data Types

Path Shape Omit Byte Masks and Shifts

```
enum gxOmitPathMask {
   gxOmitPathPositionXMask = 0xC0,
   gxOmitPathPositionYMask = 0x30,
   gxOmitPathDeltaXMask    = 0x0C,
   gxOmitPathDeltaYMask    = 0x03
};

enum gxOmitPathShift {
   gxOmitPathPositionXShift   = 6,
   gxOmitPathPositionYShift   = 4,
   gxOmitPathDeltaXShift      = 2,
   gxOmitPathDeltaYShift      = 0
};
```

Bitmap Shape Omit Byte Masks and Shifts

```
enum gxOmitBitmapMask1 {
   gxOmitBitmapImageMask      = 0xC0,
   gxOmitBitmapWidthMask      = 0x30,
   gxOmitBitmapHeightMask     = 0x0C,
   gxOmitBitmapRowBytesMask   = 0x03
};

enum gxOmitBitmapShift1 {
   gxOmitBitmapImageShift     = 6,
   gxOmitBitmapWidthShift     = 4,
   gxOmitBitmapHeightShift    = 2,
   gxOmitBitmapRowBytesShift  = 0
};
```

```
enum gxOmitBitmapMask2 {
   gxOmitBitmapPixelSizeMask  = 0xC0,
   gxOmitBitmapSpaceMask      = 0x30,
   gxOmitBitmapSetMask        = 0x0C,
   gxOmitBitmapProfileMask    = 0x03
};

enum gxOmitBitmapShift2 {
   gxOmitBitmapPixelSizeShift = 6,
   gxOmitBitmapSpaceShift     = 4,
   gxOmitBitmapSetShift       = 2,
   gxOmitBitmapProfileShift   = 0
};

enum gxOmitBitmapMask3 {
   gxOmitBitmapPositionXMask = 0xC0,
   gxOmitBitmapPositionYMask = 0x30
};

enum gxOmitBitmapShift3 {
   gxOmitBitmapPositionXShift = 6,
   gxOmitBitmapPositionYShift = 4
};
```

Bit Image Omit Byte Masks and Shifts

```
enum gxOmitBitImageMask {
   gxOmitBitImageRowBytesMask = 0xC0,
   gxOmitBitImageHeightMask   = 0x30,
   gxOmitBitImageDataMask     = 0x08
};

enum gxOmitBitImageShift {
   gxOmitBitImageRowBytesShift   = 6,
   gxOmitBitImageHeightShift     = 4,
   gxOmitBitImageDataShift       = 3
};
```

Text Shape Omit Byte Masks and Shifts

```
enum gxOmitTextMask {
   gxOmitTextCharactersMask   = 0xC0,
   gxOmitTextPositionXMask    = 0x30,
   gxOmitTextPositionYMask    = 0x0C,
   gxOmitTextDataMask         = 0x02
};

enum gxOmitTextShift {
   gxOmitTextCharactersShift  = 6,
   gxOmitTextPositionXShift   = 4,
   gxOmitTextPositionYShift   = 2,
   gxOmitTextDataShift        = 1
};
```

Glyph Shape Omit Byte Masks and Shifts

```
enum gxOmitGlyphMask1 {
   gxOmitGlyphCharactersMask  = 0xC0,
   gxOmitGlyphLengthMask      = 0x30,
   gxOmitGlyphRunNumberMask   = 0x0C,
   gxOmitGlyphOnePositionMask = 0x02,
   gxOmitGlyphDataMask        = 0x01
};

enum gxOmitGlyphShift1 {
   gxOmitGlyphCharactersShift    = 6,
   gxOmitGlyphLengthShift        = 4,
   gxOmitGlyphRunNumberShift     = 2,
   gxOmitGlyphOnePositionShift   = 1,
   gxOmitGlyphDataShift          = 0
};

enum gxOmitGlyphMask2 {
   gxOmitGlyphPositionsMask   = 0xC0,
   gxOmitGlyphAdvancesMask    = 0x20,
   gxOmitGlyphTangentsMask    = 0x18,
   gxOmitGlyphRunsMask        = 0x04,
   gxOmitGlyphStylesMask      = 0x03
};
```

```
enum gxOmitGlyphShift2 {
   gxOmitGlyphPositionsShift  = 6,
   gxOmitGlyphAdvancesShift   = 5,
   gxOmitGlyphTangentsShift   = 3,
   gxOmitGlyphRunsShift       = 2,
   gxOmitGlyphStylesShift     = 0
};
```

Layout Shape Omit Byte Masks and Shifts

```
enum gxOmitLayoutMask1 {
   gxOmitLayoutLengthMask     = 0xC0,
   gxOmitLayoutPositionXMask  = 0x30,
   gxOmitLayoutPositionYMask  = 0x0C,
   gxOmitLayoutDataMask       = 0x02
};

enum gxOmitLayoutShift1 {
   gxOmitLayoutLengthShift    = 6,
   gxOmitLayoutPositionXShift = 4,
   gxOmitLayoutPositionYShift = 2,
   gxOmitLayoutDataShift      = 1
};

enum gxOmitLayoutMask2 {
   gxOmitLayoutWidthMask   = 0xC0,
   gxOmitLayoutFlushMask   = 0x30,
   gxOmitLayoutJustMask    = 0x0C,
   gxOmitLayoutOptionsMask = 0x03
};

enum gxOmitLayoutShift2 {
   gxOmitLayoutWidthShift     = 6,
   gxOmitLayoutFlushShift     = 4,
   gxOmitLayoutJustShift      = 2,
   gxOmitLayoutOptionsShift   = 0
};
```

```
enum gxOmitLayoutMask3 {
   gxOmitLayoutStyleRunNumberMask= 0xC0,
   gxOmitLayoutLevelRunNumberMask= 0x30,
   gxOmitLayoutHasBaselineMask   = 0x08,
   gxOmitLayoutStyleRunsMask     = 0x04,
   gxOmitLayoutStylesMask        = 0x03
};

enum gxOmitLayoutShift3 {
   gxOmitLayoutStyleRunNumberShift  = 6,
   gxOmitLayoutLevelRunNumberShift  = 4,
   gxOmitLayoutHasBaselineShift     = 3,
   gxOmitLayoutStyleRunsShift       = 2,
   gxOmitLayoutStylesShift          = 0
};

enum gxOmitLayoutMask4 {
   gxOmitLayoutLevelRunsMask     = 0x80,
   gxOmitLayoutLevelsMask        = 0x40
};

enum gxOmitLayoutShift4 {
   gxOmitLayoutLevelRunsShift = 7,
   gxOmitLayoutLevelsShift    = 6
};
```

Picture Shape Omit Byte Masks and Shifts

```
enum gxOmitPictureParametersMask {
   gxOmitPictureShapeMask        = 0xC0,
   gxOmitOverrideStyleMask       = 0x30,
   gxOmitOverrideInkMask         = 0x0C,
   gxOmitOverrideTransformMask   = 0x03
};

enum gxOmitPictureParametersShift {
   gxOmitPictureShapeShift       = 0x6,
   gxOmitOverrideStyleShift      = 0x4,
   gxOmitOverrideInkShift        = 0x2,
   gxOmitOverrideTransformShift  = 0x0
};
```