

Introduction to QuickDraw GX

This chapter introduces the QuickDraw GX object-based approach to graphics programming. Any QuickDraw GX programming you do requires a basic understanding of objects and how to manipulate them. Read this chapter before reading any other chapter in this book, and before reading subsequent books in the QuickDraw GX suite, such as *Inside Macintosh: QuickDraw GX Graphics*, *Inside Macintosh: QuickDraw GX Typography*, and *Inside Macintosh: QuickDraw GX Printing*.

You can also start learning about QuickDraw GX by reading the book *QuickDraw GX Programmer's Overview*, either before or in conjunction with reading this chapter and the rest of the QuickDraw GX suite. *QuickDraw GX Programmer's Overview* introduces you to QuickDraw GX concepts through designing and building code samples.

This chapter starts by outlining the features and advantages of QuickDraw GX. It then describes

- the kinds of objects defined by QuickDraw GX
- how your application interacts with QuickDraw GX memory
- how to use objects to draw and hit-test shapes
- how to use objects to print documents
- how to program within the QuickDraw GX environment

The chapter concludes with a table and diagram summarizing QuickDraw GX objects and their properties.

What Is QuickDraw GX?

QuickDraw GX is a programming environment and toolbox for powerful two-dimensional color graphics programming. QuickDraw GX helps you create graphic and typographic objects and display them on a variety of imaging devices, including printers. The QuickDraw GX software architecture is based on objects and is compatible with, but does not require, object-oriented programming techniques.

QuickDraw GX is a large system that provides many benefits. The rest of this section summarizes some of those benefits, in terms of its three principal areas of application: color graphics, typography, and printing.

Color Graphics

QuickDraw GX is a powerful graphics engine with integrated color support, a wide range of graphics primitives, and sophisticated modes of drawing. It can manipulate images in quite general ways, leading to many useful special effects. Highlights of the graphics capabilities of QuickDraw GX include the following:

- Multiple types of graphic shapes. QuickDraw GX directly supports geometric shapes (points, lines, rectangles, polygons, curves, and paths), bitmap shapes, and picture shapes (shapes that are collections of other shapes).
- Multiple types of typographic shapes. QuickDraw GX directly supports text shapes, glyph shapes, and layout shapes, which range from simple unstyled lines of text to multilanguage, multifont text lines with sophisticated typographic features.
- Device independence. All positions and measurements in QuickDraw GX are independent of the resolution of any imaging device.
- Flexible and powerful transformations. QuickDraw GX uses mathematical mappings to easily manipulate positions, dimensions, and distortions of shapes.
- Easy stylistic variations. QuickDraw GX gives you great flexibility in setting shape characteristics such as pen width, patterns, font, and text face.
- Device-independent colors. All colors in QuickDraw GX can be defined in a device-independent way and then converted to device-specific colors on any device.
- Direct support for many color spaces, including luminance (for grayscale), RGB (for monitors), YIQ (for color video broadcast), CMYK (for printing), CIE and related device-independent color spaces (for colorimetrics).
- Automatic color matching. QuickDraw GX automatically uses color profiles and the Macintosh ColorSync utilities to guarantee that a document's colors as displayed on a monitor match as closely as possible a printed copy of the same document. If you need to, you can also manually control the color matching process.
- A sophisticated yet straightforward rendering mechanism. The mechanism allows multiple simultaneous views of a single shape, with different scales and orientations, on single or multiple devices, with simultaneous updating of all views if the shape is edited.

Compatibility With QuickDraw

QuickDraw GX does not replace the original QuickDraw architecture built into the Macintosh toolbox. An application that is not QuickDraw GX-aware is unaffected if QuickDraw GX is installed on the system. A QuickDraw GX application can also use standard QuickDraw calls and convert QuickDraw picture files into QuickDraw GX shapes. See the Macintosh environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities* for more information. ♦

The color graphics capabilities of QuickDraw GX are described both in this book and in *Inside Macintosh: QuickDraw GX Graphics*.

Typography

QuickDraw GX treats text both as text (a sequence of character codes that can be displayed and edited) and as graphics, meaning that all of the color graphics capabilities of QuickDraw GX are available for the display of text.

Each line of text can be a shape in QuickDraw GX. Using the typographic features of QuickDraw GX, you can generate and manipulate fully editable, text-related shapes with characteristics such as the following:

- **Simplicity.** Text can have a single font at a single size, with no changes in stylistic variation along the line. This type of text is most useful in dialog boxes or other situations where relatively unsophisticated string presentation is needed.
- **Flexible alignment and justification.** Text can be (1) left aligned, right aligned, or any point of alignment in between (including centered); and (2) unjustified, fully justified, or any level of justification in between.
- **Multiple styles.** Each glyph or any set of glyphs can be styled (given a font, size, or set of typographic characteristics) independently of every other glyph.
- **Independent glyph positions.** Each glyph can have any style and be positioned independently of every other glyph, so that text can be made to follow a curved path or circle.
- **Sophisticated layout.** Text lines can exhibit great typographic sophistication, with features such as kerning, tracking, shifting, ligature formation, and contextual glyph substitution.
- **Multilanguage text handling.** Text can be properly formatted in any language supported by a QuickDraw GX font, even contextual right-to-left languages such as Arabic, or languages with large character sets such as Chinese. Multiple languages, even with mixed text directions, can coexist on the same line.
- **Vertical text.** Text such as Japanese and Chinese can be written vertically, and intermixed with properly oriented vertical Roman text.

Because a line of text is a QuickDraw GX shape, you can color it, fill it with a pattern, scale it, rotate it, and transform it like any graphic shape—all the while maintaining its identity and editability as a text line. You can also use certain typographic shapes, either as-is or converted to purely geometric shapes, to perform further graphic operations with them, such as clipping, dashing, and patterning.

QuickDraw GX also provides functions that help you manipulate sets of text lines, even the most typographically sophisticated text lines, for word-processing tasks such as hit-testing and line-breaking.

Much of the text-layout sophistication of QuickDraw GX depends on information in tables in QuickDraw GX fonts, which have many features—some of which may be enabled by default—that your application can use or disable, as desired.

The typographic capabilities of QuickDraw GX are described in detail in *Inside Macintosh: QuickDraw GX Typography*.

Printing

QuickDraw GX includes an extensible, device-independent printing architecture that provides a high level of support for both users and application developers, and that makes creation of printing extensions and printer drivers fast and efficient. The printing features of QuickDraw GX include the following:

- A consistent application printing interface, regardless of the type of printer used.
- A message-based printing system. Drivers, extensions, and even applications need only respond to (override) a standard set of printing messages if they wish to add specific functionality.
- A set of printing objects that controls the printing process. Use of multiple objects means that, for example, different parts of a document can print on several printers simultaneously, or a single document can have multiple page formats for printing.
- Support for desktop printers, which are represented by icons on the computer desktop. The user can print a document by dragging it to the icon. Desktop printers support printer sharing, and you can control jobs in the print queue of a desktop printer.
- Customizable printing dialog boxes. In addition to standard print options, these dialog boxes also provide additional controls such as the ability to select a paper tray.
- The capability of creating and reading portable digital documents (PDDs). These documents can be viewed or printed on any computer that has QuickDraw GX installed, without requiring the original application or fonts with which the document was created. If QuickDraw GX is installed, any application—including those that are not QuickDraw GX-aware—can create a PDD.
- Fast development of printing extensions that can work with any printer driver and any application, to extend the printing capabilities available to the user.
- Fast development of printer drivers.

To implement the basic printing capabilities of QuickDraw GX, your application need provide only a small amount of code, which executes in response to a few menu items and a single printing message. With additional developmental effort, you can provide highly customized capabilities. (Even if your application implements no QuickDraw GX printing features at all, its users receive the benefit of desktop printers.)

The application printing interface to QuickDraw GX is described in *Inside Macintosh: QuickDraw GX Printing*; the interface for printing extensions and printer drivers is described in *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

What QuickDraw GX Is Not

QuickDraw GX is a powerful system, but it does have certain limitations. If your graphic programming needs are outside of the capabilities of QuickDraw GX, you may wish to implement them yourself or—where possible—use the built-in capabilities of the platform on which your application runs. For example QuickDraw GX does not provide explicit support for

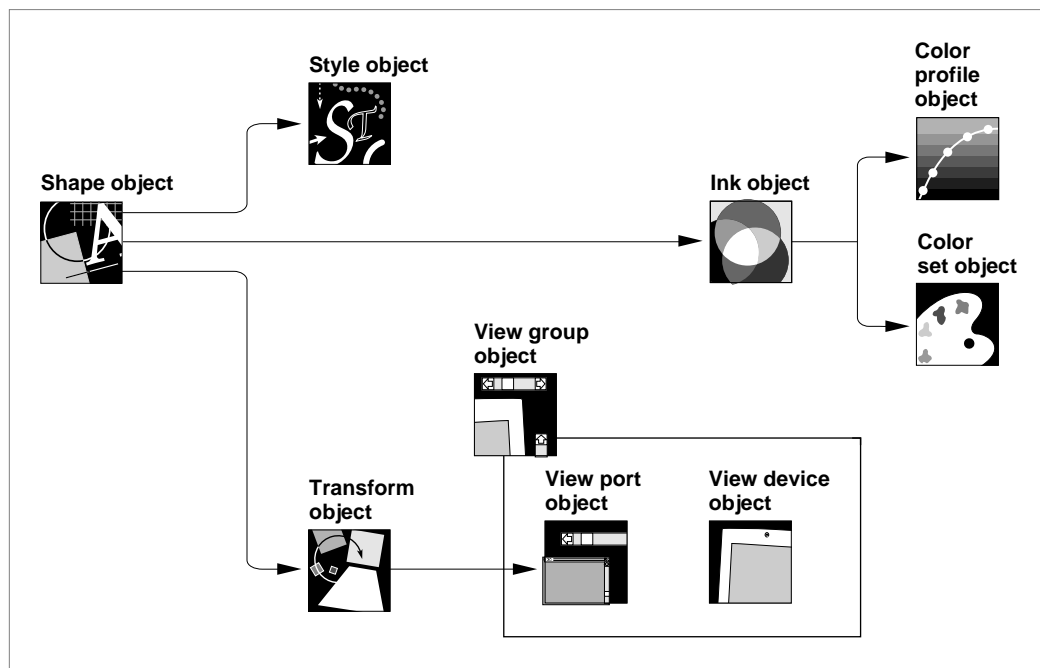
- application-definable object methods
- a floating-point interface
- multiple colors or gradient fills in shapes (other than bitmaps)
- planar-pixel devices
- 3-D rendering
- cubic curves (but conversion library code is available)
- formatting of text units greater than a line, such as paragraphs
- tabs in text
- anti-aliasing (other than alpha-channel support in bitmaps)
- palette management (handled by system software on the Macintosh)
- cursor management (handled by system software on the Macintosh)

Also, on the Macintosh, QuickDraw GX does not completely replace either QuickDraw or the Window Manager for drawing, and it does not completely replace the Script Manager and international resources for non-Roman text-handling. QuickDraw GX extends the capabilities of these managers, but in some instances you still need to use their functions.

QuickDraw GX Objects

With QuickDraw GX you create and draw objects. Fundamental graphic shapes, such as rectangles and curves, are objects. Lines of text are objects. Other pieces of information, such as the color of a shape or the font used to draw a letter, are also kept in objects. Fonts are objects. Even the information that is used to describe the printing characteristics of a document is kept in objects.

Figure 1-1 names some of the common QuickDraw GX objects and shows their relationships, in terms of which objects use the information in which other objects. This chapter, this book, and much of the rest of the *Inside Macintosh* QuickDraw GX suite describe what these and other QuickDraw GX objects are and how to use them.

Figure 1-1 Several QuickDraw GX objects

How QuickDraw GX Defines Objects

Objects are specialized data structures. Some of the data structures used by operating systems such as the Macintosh Operating System are public—that is, your application can manipulate the values of their fields directly. Many of the data structures used by QuickDraw GX, on the other hand, are not public. These private data structures are called *objects* and the accessible pieces of information inside them are called *properties*. Your application creates and modifies objects to perform tasks, but it may not manipulate object properties directly. Instead, QuickDraw GX provides functions that manipulate them for you.

QuickDraw GX does not provide pointers or handles for you to locate objects. Instead, it provides reference values. To allow type checking in C and Pascal, QuickDraw GX defines references as pointers to structures, although the reference is *not* guaranteed to point to anything. For example, a shape object is identified by a shape *reference*:

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

The contents of the structure are private. To obtain information about an object, you must send its reference as a parameter to a QuickDraw GX function.

Introduction to QuickDraw GX

When you create an object, you call a *GXNewObject* function that returns a reference to the object. Conversely, you can dispose of an object you no longer need by passing its reference in a call to *GXDisposeObject*. For example, you can create a picture shape object by calling the *GXNewShape* function with a parameter that specifies that you want the shape to be a picture type:

```
myShape = GXNewShape(gxPictureType);
```

In this example, *myShape* is a reference to the shape object, returned by the function. When you are finished with the object, you dispose of it like this:

```
GXDisposeShape(myShape);
```

QuickDraw GX objects exist in a memory area (QuickDraw GX memory) that is separate from the application's memory. For more information on QuickDraw GX memory, see "Objects and Memory" beginning on page 1-18.

QuickDraw GX defines its objects in a device-independent manner. Because of that, and because many of its data structures are private, the QuickDraw GX software and the hardware on which it runs can evolve without disrupting existing applications.

Advantages of an Object-Based Structure

QuickDraw GX is currently implemented in the C programming language, which is not in itself object-oriented. Nevertheless, using QuickDraw GX gives you some of the fundamental programming advantages available with object-based systems.

QuickDraw GX objects are private. You do not usually have direct access to the internal data in a QuickDraw GX object; you instead make function calls to manipulate the information. This information hiding means that objects behave more consistently, unwanted side effects are minimized, and QuickDraw GX itself can take care of housekeeping tasks like tracking the current number of users of an object. It also means that QuickDraw GX can locate objects in memory managed by a graphics accelerator—memory that is not necessarily accessible to your application.

By analogy with the polymorphism of some object-oriented systems, QuickDraw GX functions are organized so that a single function can apply to many types of objects. For example, a single drawing command (*GXDrawShape*) draws any QuickDraw GX shape, from a point to a curve to a bitmap to a line of text. Furthermore, there are many classes of calls that, while defined individually for each kind of object they apply to (in order to facilitate type-checking in Pascal and C), are completely parallel in function and in syntax. For example, the functions *GXGetShapeTags*, *GXGetStyleTags*, and *GXGetInkTags* take the same parameter (an object reference) and perform the same task (return a list of associated tag objects), but each for a different kind of object.

Introduction to QuickDraw GX

QuickDraw GX objects can be shared. To save duplication and prevent the accumulation of excessive numbers of objects in memory, QuickDraw GX allows multiple references to a single object. QuickDraw GX tracks the number of references to an object. When you are finished with an object, you dispose of it; QuickDraw GX then makes sure that the object is not being used for any other purpose before actually deleting it from memory.

Creating a QuickDraw GX object is somewhat like instantiating a class in an object-oriented system. When you first create a QuickDraw GX object it typically has default values that you can use or change to suit your needs.

Object-manipulation functions are mostly consistent across all objects; categories include `GXNewObject` (makes a new object), `GXDisposeObject` (deletes the object), `GXCopyToObject` (copies an object), `GXEqualObject` (tests two objects for equality), and `GXCloneObject` (makes a shared reference). Object-editing functions are similarly consistent, and include `GXGetObjectProperty` (to retrieve values) and `GXSetObjectProperty` (to assign values). By combining `GXGetObject` and `GXSetObject` calls with index values and ranges, you can insert, delete, and replace all or parts of arrays of values within an object.

The QuickDraw GX environment provides other consistencies to make programming tasks more straightforward. Many are listed in the section “Programming Conventions and Consistencies” beginning on page 1-41.

Kinds of QuickDraw GX Objects

There are about a dozen different kinds of QuickDraw GX objects that you can use, beginning with the most fundamental object, the shape. Figure 1-1 on page 1-8 shows some of those objects and how they relate to each other; this section describes them and others.

Shape Objects

A *shape* is something that you can draw. Besides drawing it, you can also measure, parse, move, rotate, distort, check for intersection and union, make bold, simplify, and otherwise manipulate it. The fundamental purpose of QuickDraw GX is to create, manipulate, and draw shapes.

A shape consists of a *shape object* and three other associated objects (style, ink, and transform). A shape object consists of a *geometry* of a certain *shape type* (such as a line, rectangle, bitmap, or text) and information about how the geometry is framed or filled when drawn. A shape also has attributes, such as whether it should be stored in accelerator-card memory, if present. It also has references to its other three related objects.

Shapes and shape objects in general are discussed in the chapter “Shape Objects” in this book. More specifically, however, shapes are divided into types. There are two basic categories of shape type: graphic and typographic.

Graphic Shapes

Graphic shapes include geometric shapes, bitmap shapes, and picture shapes:

- Geometric shapes are the building blocks for drawing. Geometric shapes, alone or in combination, make up the graphic elements supported by drawing programs. The defined types of geometric shapes are point, line, rectangle, curve, polygon, and path. There are two other special types of geometric shapes: empty and full. An empty shape has no extent, and a full shape has the maximum possible extent.
- Bitmap shapes contain bit images or pixel images. QuickDraw GX bitmaps can be black and white, grayscale, or color.
- Picture shapes are collections of other QuickDraw GX shapes. Picture shapes can contain other picture shapes, in a hierarchy. Picture shapes allow you to override some characteristics of the contained shapes.

Graphic shapes are described further in *Inside Macintosh: QuickDraw GX Graphics*. That book also describes functions for performing geometric operations, such as measurement, simplification, and constructive geometry, on graphic and typographic shapes.

Typographic Shapes

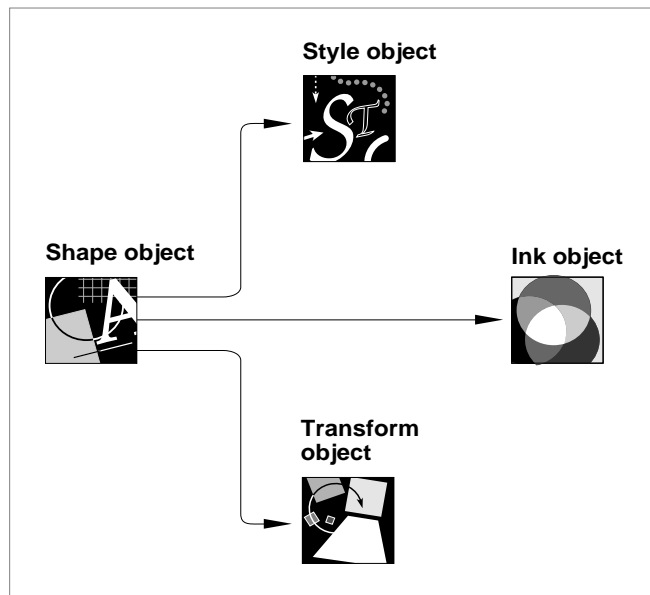
Typographic shapes represent text items—individual glyphs, collections of glyphs, or lines of text. The geometry of a typographic shape contains the text characters or glyphs of the shape, plus other information. There are three kinds of typographic shapes:

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

Typographic shapes are described further in *Inside Macintosh: QuickDraw GX Typography*.

Supporting Objects

Several other QuickDraw GX objects exist in support of shape objects. They are either directly or indirectly referenced by the shape object whose behavior they affect. Figure 1-2 shows the three objects that are directly referenced by a shape object; Figure 1-1 on page 1-8 includes these objects as well as additional objects referenced indirectly by the shape object.

Figure 1-2 A shape object and its referenced objects

Style Object

A *style object* describes certain characteristics affecting how a shape is drawn. For geometric shapes, this includes information such as the thickness of the pen, the joins between line segments, and any dash or pattern to apply to the shape. For typographic shapes, it includes information such as the font, text size, and typeface of the text. For layout shapes in particular, it includes information such as kerning behavior and font-feature selection.

Style objects in general are described in the chapter “Style Objects” in this book. Style objects used by graphic shapes are described in the geometric styles chapter of *Inside Macintosh: QuickDraw GX Graphics*; style objects used by typographic shapes are described in the typographic styles chapter of *Inside Macintosh: QuickDraw GX Typography*.

Ink Object

An *ink object* describes a shape’s color and its transfer mode—how that color is applied when the shape is drawn. Inks support many different kinds of color specification, and many different transfer modes.

Ink objects are described in the chapter “Ink Objects” in this book.

Transform Object

A *transform object* describes the clip and mapping applied to a shape when it is drawn. The clip limits the extent of the shape; it can be described by any shape geometry, and QuickDraw GX provides constructive geometry functions with which you can easily manipulate clips by combining them with other shapes. The mapping is a 3×3 matrix that defines translation, scaling, skewing, rotation, or perspective. Transforms also describe information used for hit-testing a shape and its parts. Transforms have references to one or more view ports, objects that describe where the shapes are drawn.

Transform objects are described in the chapter “Transform Objects” in this book.

Color Set Object and Color Profile Object

A *color set object* is like a color table; it contains an indexed set of colors. Color sets are used when colors are specified by index instead of by direct color value. Bitmaps commonly use color sets.

A *color profile object* contains color matching information. The information in a color profile can be used to convert device-specific colors to device-independent colors, to provide the most faithful reproduction of colors on different devices. QuickDraw GX can automatically perform color matching with available color profiles whenever it draws.

Color sets and color profiles are described in the chapter “Color and Color-Related Objects” in this book.

View Port Object, View Device Object, and View Group Object

A *view port object* is the location into which an application draws a shape. A view port object has a clip and a mapping that define a window (or a part of a window, such as a window pane). View ports can be arranged in a hierarchy.

A *view device object* typically describes a physical display device such as a monitor or printer (or an area of memory for offscreen drawing). It has a mapping, a clip, and a bitmap that describe the view device’s position, dimensions, pixel depth and colors, and color profile.

A *view group object* describes an imaging world, the global space in which view ports and view devices are located. Within a view group, view ports and view devices can overlap each other in any combination; the intersection of each view port with a view device determines what is actually drawn on that device.

View ports, view devices, and view groups are described in the chapter “View-Related Objects” in this book.

Tag Object

A *tag object* is a general container for information that an application wants to add to a QuickDraw GX object. Tag objects can have anything in them, from labels to alternate drawing instructions to anything else you feel is useful. You can attach a tag object to the tag list of most other kinds of objects (except other tag objects).

Tag objects are described in the chapter “Tag Objects” in this book.

Font Object

A **font object** is the QuickDraw GX representation of an installed font. A font object contains information about the font's names, encodings, font variations, and other tables. See the fonts chapter of *Inside Macintosh: QuickDraw GX Typography* for more information.

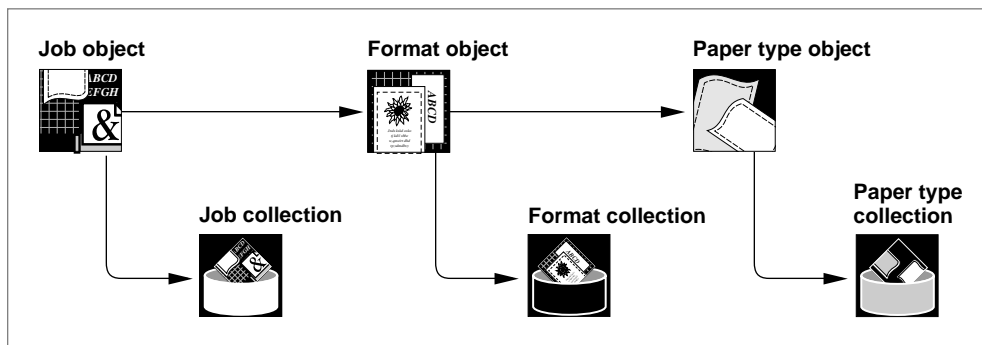
Graphics Client Object

A graphics client is the object representation of the QuickDraw GX memory allocated for an application, which is separate from the application's own memory. A graphics client has no accessible properties, and in most cases your application never explicitly creates one. See the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities* for more information.

Printing Objects

One category of QuickDraw GX objects exists to support printing. The printing objects include those shown in Figure 1-3 plus several others. Figure 1-3 shows the three principal QuickDraw GX printing objects (job, format, and paper-type), plus the three collection objects they use.

Figure 1-3 Printing objects



Note

Printing objects are different in some aspects from other QuickDraw GX objects. Most importantly, they exist in application memory instead of QuickDraw GX memory; this affects their behavior in several ways, as noted in later sections of this chapter. ♦

Job Object, Format Object, and Paper-Type Object

The **job object** is the primary holder of printing information for a document. Every printable document has a job object associated with it. The job object specifies information such as the number of copies and the page range for printing, and includes references to one or more format objects and two printer objects (one for formatting and one for current output).

The *format object* specifies information such as scaling and page dimensions for the document, and includes a reference to a paper-type object.

The *paper-type object* specifies information such as a paper-type name (such as “US Letter”) and the physical dimensions of the paper.

See the core printing features chapter of *Inside Macintosh: QuickDraw GX Printing* for more information.

Collection Objects

The job object, format object, and paper type object also include references to *collection objects*, which are objects managed by the Collection Manager, a part of system software provided with QuickDraw GX. Collection objects can contain any type of data; for printing, they hold additional useful information, such as specifications for halftoning, that is not in the printing objects. The Collection Manager is described in the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Printer Object

The *printer object* is another printing object. It represents a physical printer and includes a name and type, a driver name and type, and a reference to one or more view device objects that describe the characteristics of the printer the application draws to when printing. See the advanced printing features chapter of *Inside Macintosh: QuickDraw GX Printing* for more information.

Print-File Object

A *print-file object* is a printing object that represents a print file, the file that is the printable representation of a document. When it prints a document, QuickDraw GX first creates a print file, and then uses that print file to create an image on a printer. See the advanced printing features chapter of *Inside Macintosh: QuickDraw GX Printing* for more information.

Object Properties

The accessible information in an object is its set of properties. Object properties are like the fields of a structure, except that they are accessible only through function calls; you cannot read them directly. Each property consists of a value or a list of values; the definition of the property determines what it contains. For example, the shape type property of a shape object contains a value, such as `gxRectangleType`, that describes the type of shape that it is.

For most properties, QuickDraw GX provides `GXGetObjectProperty` and `GXSetObjectProperty` functions that allow you to get or set each accessible part of the object. For example, the following statement returns a shape object’s type into the `myShapeType` variable:

```
myShapeType = GXGetShapeType(myShape);
```

Introduction to QuickDraw GX

Figure 1-13 on page 1-49 lists the accessible properties of the principal QuickDraw GX objects other than printing objects. Note that, because they are properties and not fields, their order in Figure 1-13 is arbitrary. The properties are explained in more detail in the chapter that describes the object.

Some object properties are common to most kinds of objects. For example, many objects have properties that are simply references to other objects. In addition, many objects have attributes, an owner count, and a tag list. These four kinds of common properties are summarized in this section.

References

Some properties consist of references to other objects. These references define a relationship between the objects; the properties of the referenced object are like an extension to the properties of the object containing the reference. For example, Figure 1-2 on page 1-12 shows three objects referenced by a shape object: a style object, an ink object, and a transform object. Those three objects' properties affect how the shape that references them is drawn; the ink object, for example, defines the color of the shape.

Many objects contain references to other objects. Some object properties are individual references, whereas other properties are arrays, or lists, of references to several objects. The advantages of using object references are discussed in the section "Sharing and Multiple Object References" beginning on page 1-19.

Note

In illustrations of object properties throughout the QuickDraw GX documentation, properties that are object references (or lists of object references) are represented in italics. See, for example, how the style, ink and transform properties of the shape object are represented in Figure 1-13 on page 1-49. ♦

Attributes

Some objects have an attributes property, which is a group of flags that you use to modify the behavior of the object. In shapes, for example, these flags allow you to specify—among other things—how QuickDraw GX stores the shape object and how editing operations affect the shape object. In view ports, as another example, these flags allow you to specify behavior such as whether or not to perform color matching when drawing.

Owner Count

For objects that are shared, this property indicates how many references to the object exist. For example, when you create a new shape object, QuickDraw GX sets the owner count of the new shape to 1. If you add that shape to a picture, QuickDraw GX increments the shape's owner count by 1. If you dispose of the picture, QuickDraw GX decrements the shape's owner count by 1. Whenever the owner count of a shared object reaches 0, the object is deleted and its memory released.

Owner counts are discussed further in the section "Sharing and Multiple Object References" beginning on page 1-19.

Tag List

This property is an array of references to custom information stored in tag objects. Tag objects are discussed further in the section “Adding Custom Behavior With Tag Objects,” on this page.

Default Objects and Default Properties

QuickDraw GX provides default versions for all types of shape objects, and default values for the properties of other objects such as styles, inks, transforms, color sets, and color profiles. Therefore, when you create an object with a `GXNewObject` call, its properties are already set to match the default. For example, the default rectangle shape object has an owner count of 1, a solid shape fill, corners at locations (0.0, 0.0) and (0.0, 0.0), and a reference to the default ink object. If you want the new shape to have different dimensions or to reference a different ink object, you can change those properties after creating the shape.

The default shape objects are unique among QuickDraw GX default objects in that you can change them. If you want every new shape of a certain type to start off with a particular set of properties, you can change the properties of the default shape for that shape type, and every new shape of that type that you create will have the new properties.

You cannot change the default for most other objects. However, you can effectively change the default for any object that is referenced directly or indirectly by a shape object. For example, you can effectively create a new default ink object by first creating a version of the ink object that has the properties you want, and then altering all default shape objects to reference that ink object instead of the default ink object.

For objects for which there is no changeable default, there are nevertheless default values that are applied to the object when it is first created.

Default color sets and color profiles

Color sets have changeable default versions, but they function differently than default shapes. You can define a color set to be the default associated with bitmaps of a given pixel depth. However, when you create a color set using the `GXNewColorSet` function, it has specific properties that are unaffected by any previous definitions of defaults.

There is a single default color profile, applied by QuickDraw GX to colors that do not have an attached profile. The default profile is not directly changeable. ♦

Adding Custom Behavior With Tag Objects

A tag object is a special kind of object whose purpose is to allow any type of application-defined information to be attached to a QuickDraw GX object. An object such as a shape or transform can be “tagged” with data or code that provides extra information about it or allows you to alter its behavior in specific situations.

Introduction to QuickDraw GX

You can, for example, attach identifying strings to objects with tags. As another example, you can alter the way an object is displayed on a particular imaging device (such as a PostScript device) by attaching a tag to it that contains imaging commands specific to that device.

A tag object is attached to its associated object by means of a *tag list*, a property that most QuickDraw GX objects have. A tag list is an array of references to the tag objects attached to an object. Objects can thus have more than one attached tag object.

Because tags are QuickDraw GX objects, they can be shared. Like other QuickDraw GX objects, tags are accessible from objects in accelerator memory, they can be unloaded to disk and reloaded automatically, and they can be flattened (see “External Storage of Objects: Flattening and Unflattening” on page 1-23). See the chapter “Tag Objects” in this book for more information.

Objects and Memory

Objects are structures in memory. The way QuickDraw GX manages memory is central to its object orientation and to the advantages it provides you. QuickDraw GX has its own memory, and gives you access to it only in restricted situations.

Application Memory and QuickDraw GX Memory

When you program with QuickDraw GX, you are concerned with at least two separate memory heaps: the *application heap*, which holds your code and data structures, and a part of QuickDraw GX memory called the *graphics client heap*, which holds the objects you create with Quickdraw GX. As an application, you allocate variables and execute in application memory. You can directly access any data structures in that heap. Much of Macintosh system software, including the toolbox, can affect the application heap, sometimes in unwanted ways (as during memory compaction).

QuickDraw GX rarely uses the application heap (except for storing printing-related objects). It allocates its objects, structures, and variables in the graphics client heap. QuickDraw GX memory is private; you cannot directly access the contents of the graphics client heap except under special conditions. The graphics client heap does not even have to be in the same physical address space as the application heap. For example, QuickDraw GX can execute from and store objects in the memory on a graphics accelerator card.

QuickDraw GX objects are private because they are in private memory. That means you must make QuickDraw GX calls to access objects and their information, but it also means that you can make almost any call without worrying that it might move application memory.

Typically, your application manages its own structures in the application heap, and makes function calls to obtain or change the contents of the graphics client heap. For example when you call a `GXGetObjectProperty` function, QuickDraw GX places a copy of the contents of an object's property in your application's heap. If you modify the information, you can then call a `GXSetObjectProperty` function to copy the new values from your application's heap back into the object in the graphics client heap.

If you are a Macintosh programmer, remember that QuickDraw GX memory is completely separate, and you needn't be concerned about its location or contents. Macintosh Memory Manager functions cannot allocate, resize, or determine the size of any QuickDraw GX object. To manage its memory, QuickDraw GX has its own internal memory manager and memory management functions. See the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities* for more information. See *Inside Macintosh: Memory* for information on the Macintosh Memory Manager.

The QuickDraw GX memory manager may move objects, unload them to disk if necessary, and reload them when they are needed again. To reference and use an object, you needn't be concerned with or even know whether it is in a loaded or unloaded state. QuickDraw GX automatically loads any unloaded object when it is needed, even if that means unloading another object to make room. See "Automatic Loading and Unloading of Objects" on page 1-21 for more information.

Sharing and Multiple Object References

Object-based systems can use large amounts of memory, especially when an application needs to create and use many objects. To minimize redundancy and excess memory use, QuickDraw GX supports the sharing of objects.

For example, you may want to create a set of shape objects that are of different sizes and geometries, but that all have the same color and are drawn with the same transfer mode. You can create a single ink object with the desired color and transfer mode that all the shapes can reference, without having to create a separate ink object for each shape. In that situation there is one reference to the ink object for each shape that uses it.

Alternatively, your application can create data structures that contain object references, and two or more structures can contain references to the same object. For example, different palette structures can contain references to the same color set object that defines the palette colors. In that situation there is one reference to the color set object for each palette that uses it.

Sharing is not the same as making a copy. No matter how many references there are to an object, it is still only a single object. When you change any aspect of a shared object, those changes are reflected in every other object or data structure that references that object.

Introduction to QuickDraw GX

Object sharing provides at least three advantages:

- It reduces memory use. Some objects that are used by many other objects are quite large. For example, a font, which can be a very large object, can be used by several different styles. And each style can be used by several text shapes.
- It gives uniform behavior. For example, several shapes can share the same transform object, which causes each shape to be drawn in a specific relationship to each other, scaled and rotated in the same way, and so on.
- It allows quick and efficient changes to the characteristics of multiple objects that share the same reference. For example, if several shape objects reference the same ink object, you need only change the color in the ink object to change the color of all shapes that reference the same ink.

Owner Count

The current number of references to an object is called its *owner count*. QuickDraw GX tracks and manages owner counts for you, so in most cases you needn't worry about how many references there are to an object and whether or not to delete it from memory when you no longer need it in a given context.

When you first create an object (with a call such as `GXNewStyle`), QuickDraw GX gives it an initial owner count of 1. Whenever you attach that object to another object (with a call such as `GXSetShapeStyle`), QuickDraw GX does not duplicate it; instead, it increases the object's owner count by 1. Whenever you delete that object (with a call such as `GXDisposeStyle`) or any object that references it (with a call such as `GXDisposeShape`), QuickDraw GX decreases its owner count by 1.

QuickDraw GX uses the owner count to determine when an object is no longer needed and can be deleted. If at any time the object's owner count decreases to zero, QuickDraw GX deletes it from QuickDraw GX memory. As far as your application is concerned, you create and dispose of objects as you wish, and let QuickDraw GX decide when to actually remove them from memory.

There can be cases, however, in which the owner count would normally become 0 but you do not want the object to be deleted. In those cases, you can increase owner count with the cloning capability of QuickDraw GX, described next.

Cloning

Although QuickDraw GX can correctly track owner counts as objects are created, disposed of, and referenced from other objects, it cannot know how many references to a given object exist in variables and data structures that you have created. In these situations, it is up to you to manage the owner counts of the objects that you use. Also, you may want to preserve a reference to an object that QuickDraw GX disposes of when it disposes of or modifies another object. In such a case, you can make sure the owner count of an object correctly reflects the number of references to it by *cloning* the object, which means increasing its owner count.

For example, if you create a color set object, it has an owner count of 1. If you dispose of that color set, its owner count becomes zero and it is deleted by QuickDraw GX, as it should be. On the other hand, if you assign a new ink object to a shape, that shape's original ink object is disposed of and the owner count of the new ink object is increased by 1. If you had wanted to maintain a reference to the shape's original ink object, you could have cloned that ink before assigning the new ink to the shape. The original ink's owner count would remain above zero, and it would therefore not be deleted.

As another example, you may temporarily change the style object assigned to a shape, intending to restore that style to the shape eventually. When you assign the new style object, QuickDraw GX decrements the original style object's owner count because it is no longer used by the shape. If the original style is not used by another object, its owner count would become 0 and QuickDraw GX would delete it. To prevent that from occurring, you can clone the original style object before assigning the new one.

QuickDraw GX cannot determine when you are finished with an object once it is cloned. If you clone an object, you are responsible for disposing of it when it is no longer needed.

Some Objects Cannot Be Cloned

Some objects have no owner count because they need to be able to be deleted even when valid references to them remain. View-related objects (view ports, view devices, and view groups) and fonts are examples of such shared objects that cannot be cloned. For example, suppose a transform object references a particular view port object associated with a window. When the application closes the window, it disposes of the view port. The view port object is deleted, even though a valid reference to it still remains in the transform object. (Subsequent drawing to that view port reference has no effect; QuickDraw GX ignores references to a view-related object that does not exist.) ♦

Automatic Loading and Unloading of Objects

Another way that QuickDraw GX minimizes memory requirements is by moving objects back and forth between memory and external storage as needed. If QuickDraw GX needs additional memory to create new objects, it can unload objects that are already in memory. When unloaded, an object is moved from computer memory to temporary private storage on disk. When loaded, that object is restored to normal object form in memory.

Typically, QuickDraw GX unloads objects that have not been accessed recently before unloading objects that your application has been using frequently. Also, for shape objects, QuickDraw GX provides flags that you can set to notify QuickDraw GX that you want it to unload a given shape before all others, or unload it after all others, when more memory is needed.

To reference and use an object, you needn't be concerned with or even know whether it is in a loaded or unloaded state. QuickDraw GX automatically loads any unloaded object when it is needed, even if that means unloading another object to make room.

Introduction to QuickDraw GX

For some purposes, such as measuring the storage size of an object, you may need to have the object in memory. Conversely, in other situations you may wish to allow an object to leave memory temporarily, to make more room in the QuickDraw GX heap. QuickDraw GX provides functions (such as `GXLoadShape` and `GXUnloadShape`) that you can use to explicitly load or unload an object.

The `GXLoadShape` and `GXUnloadShape` functions, and other loading and unloading calls, are described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. The flags that affect the loading and unloading priority for shapes are described under shape attributes in the chapter “Shape Objects” in this book.

Direct Access to Object Structure: Locking and Unlocking

Normally, to modify a property of an object takes three steps. First, you make a function call to obtain a copy of the information in application memory. Then you modify the information. Finally, you make another function call to place that information back into the object in QuickDraw GX memory.

As a convenience, QuickDraw GX allows you to directly access parts of certain objects in QuickDraw GX memory in three specific situations: you can manipulate the geometric structure of a shape object, you can manipulate the profile data of a color profile object, and you can manipulate the contents of a tag object, without first having to work on copies of the data in application memory.

This direct manipulation is convenient, especially if you want to avoid copying large amounts of information, but it has a price. You must first lock the item you are accessing, so that it cannot be moved while you are working on it. When you have finished your alterations, you must be sure to unlock the item so that QuickDraw GX is free to relocate it. In the case of shape geometry, you must then make an additional call to QuickDraw GX to notify it that you have changed the shape.

Another drawback is that you cannot change the size of the item you are manipulating. If you need to make a shape’s geometry or a tag’s contents larger or smaller, you need to access the information in the normal way, through QuickDraw GX functions.

Remember also that locking an object fragments the QuickDraw GX heap, which can result in lower performance and possibly an error condition. Furthermore, in low-memory conditions, QuickDraw GX can actually unlock locked objects and move them if it needs to.

For information about locking shape objects, see the chapter “Shape Objects” in this book. For information about locking color profile objects, see the chapter “Colors and Color-Related Objects” in this book. For information about locking tag objects, see the chapter “Tag Objects” in this book.

External Storage of Objects: Flattening and Unflattening

QuickDraw GX objects exist (as objects) only in memory. You must convert a QuickDraw GX shape (a shape object and its referenced objects) into an equivalent compressed description in order to save it to external storage, transmit it across a network, or store it in the Clipboard. This process of converting objects to a compressed format that is no longer object-based is called *flattening*. The flattened information is a stream-based description with a public format, so that applications can share the data and reconstruct the objects from which the flattened stream was generated.

The data of flattened objects follows the format defined in the stream format chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. To reconstruct a shape's object-based description from its flattened stream, you can manually create and initialize a set of objects based on the information in the stream, or—if QuickDraw GX is available—you can use QuickDraw GX functions to do it automatically.

Printing objects are also flattened and unflattened as the documents they are associated with are closed and reopened. For more information, see the core printing features chapter of *Inside Macintosh: QuickDraw GX Printing*.

Portable digital documents (PDDs) are specialized versions of print files, which are the flattened versions of documents sent to printers. For more information, see “Printing With QuickDraw GX” beginning on page 1-34, and the advanced printing features chapter of *Inside Macintosh: QuickDraw GX Printing*.

Fonts are represented in QuickDraw GX as font objects, which are flattened for transmission to printers or for external storage. A flattened font's format, however, is not related to the QuickDraw GX stream format. For more information, see the fonts chapter of *Inside Macintosh: QuickDraw GX Typography*.

Drawing and Hit-Testing Shapes

Ultimately, you need QuickDraw GX to draw the shapes that you create with it, and you may also need to respond to user manipulation of those drawn shapes. For that reason, QuickDraw GX provides several drawing functions and several kinds of hit-testing capabilities. This section summarizes the QuickDraw GX drawing process and the QuickDraw GX approach to hit-testing.

Drawing

Drawing is the process of converting the internal representation of a shape into an image on an output device. As noted in Figure 1-2 on page 1-12, a QuickDraw GX shape consists of several other objects in addition to a shape object. When you draw a shape, QuickDraw GX uses information from those objects and others to control how the shape is rendered. It uses the information in this order:

- the geometry of the shape object
- stylistic and color information from the style object and ink object
- clipping and mapping information from the transform object
- mapping and clipping information from one or more view port objects
- mapping and clipping information from one or more view device objects

Drawing starts with geometry, a property of every shape object. The geometry defines the intrinsic dimensions of the shape. Those dimensions can then be modified, in several stages, until the rendered image appears on the screen or printer. The rest of this section describes in more detail how a shape's geometry is transformed as it passes through the drawing steps.

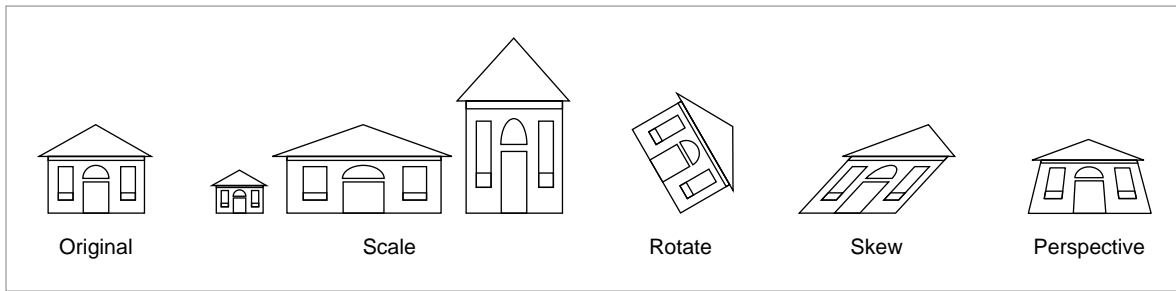
Mapping and Clipping

Mapping and clipping are two of the principal modifications a shape undergoes as it is prepared for drawing, and each occurs at several steps along the way.

A *mapping* is a 3×3 matrix that performs a mathematical transformation on a set of two-dimensional points, such as the geometry of a shape. Given any shape, you can use a mapping to control

- translating, or moving, the shape from one (x, y) location to another
- scaling the shape in the x-direction, y-direction, or both directions
- rotating the shape around any point
- skewing the shape
- changing the perspective of the shape

Figure 1-4 shows examples of the effects of mapping.

Figure 1-4 Effects of mapping

The transform object, the view port object, and the view device object each has a mapping as a property. Each object's mapping can affect the location, orientation, scale, and other distortion of the shape as it evolves from geometry to rendered image (described under "The Drawing Sequence: Coordinate Conversion" beginning on page 1-28). Mappings are described more fully in the chapter "Transform Objects" in this book, and in the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Clipping is the restriction of the visible part of a shape to a specific area. The *clip* is the specific description of that visible area. Clips are often rectangles or similar simple shapes, although QuickDraw GX permits clipping to any definable shape geometry (rectangle, polygon, path, and so on), which allows for very sophisticated clipping effects. Clips can even be glyph shapes and one-bit-per-pixel bitmaps. For the rules and restrictions on clips, see the chapter "Transform Objects" in this book.

The transform object, the view port object, and the view device object each has a clip as a property. Each object's clip is applied at a specific point during the preparation of the shape for drawing (described under "The Drawing Sequence: Coordinate Conversion" beginning on page 1-28). Each further restricts the part of the shape that will ultimately be visible.

View-Related Objects

The transform object associated with each QuickDraw GX shape contains a reference to one or more view port objects. When you draw the shape, QuickDraw GX uses that view port reference to determine at what position on which physical device or devices to draw the shape. To do that requires that the view port and two other view-related objects, the view group and view device, interact as follows:

Introduction to QuickDraw GX

- A ***view port*** object represents a drawing environment. A view port is analogous to a porthole on a ship. The view port has a mapping that defines the scale, orientation, and location of the porthole, and a clip that prevents anything beyond the edges of the porthole from being drawn. If you think of a view port as analogous to a Macintosh graphics port, the view port mapping defines the location (in QuickDraw global coordinates) of the port on the screen, and the clip defines the visible region of the port. Unlike graphics ports, however, view ports are device independent, and their mappings control much more than location: they can also define the scaling, rotation, skewing, and other distortion of shapes drawn in the view port.
- A ***view device*** object typically represents an actual, physical output device such as a monitor or printer. It, too, has a mapping and a clip that define its location and its visible (drawable) area. You can think of a view device as analogous to the Macintosh screen, in which case the mapping defines the location of the screen origin (and the size of the pixels too), and the clip defines the screen bounding rectangle. When a shape is drawn, it appears on a view device if the shape's view port intersects the view device. The object that controls the relative positions of view ports and view devices is the view group.
- A ***view group*** object represents a coordinate plane that provides dimensions and relative positions for view ports and view devices. A view group's coordinates have a specific dimension (unit distance is 1 point, or 1/72 inch). For all view devices that represent actual physical devices, QuickDraw GX defines their locations in the onscreen view group's coordinate plane. Your application then defines the locations of view ports on that plane, and thus controls whether or not the view ports are visible on the view devices. A view group is equivalent to the QuickDraw coordinate plane (or to an offscreen graphics world) on the Macintosh, and view group coordinates are analogous to QuickDraw global coordinates. However, unlike with QuickDraw, QuickDraw GX global coordinates have a specific dimension and are device independent.

Figure 1-5 shows schematically how these objects interact as a shape is drawn. A shape geometry that defines a vase, a gray color defined in the ink object, a thick pen width defined in the style object, and a scaling in the transform object's mapping combine to make an elongated image of the vase. A portion of the vase appears on screen, where the clips of the view port and view device overlap.

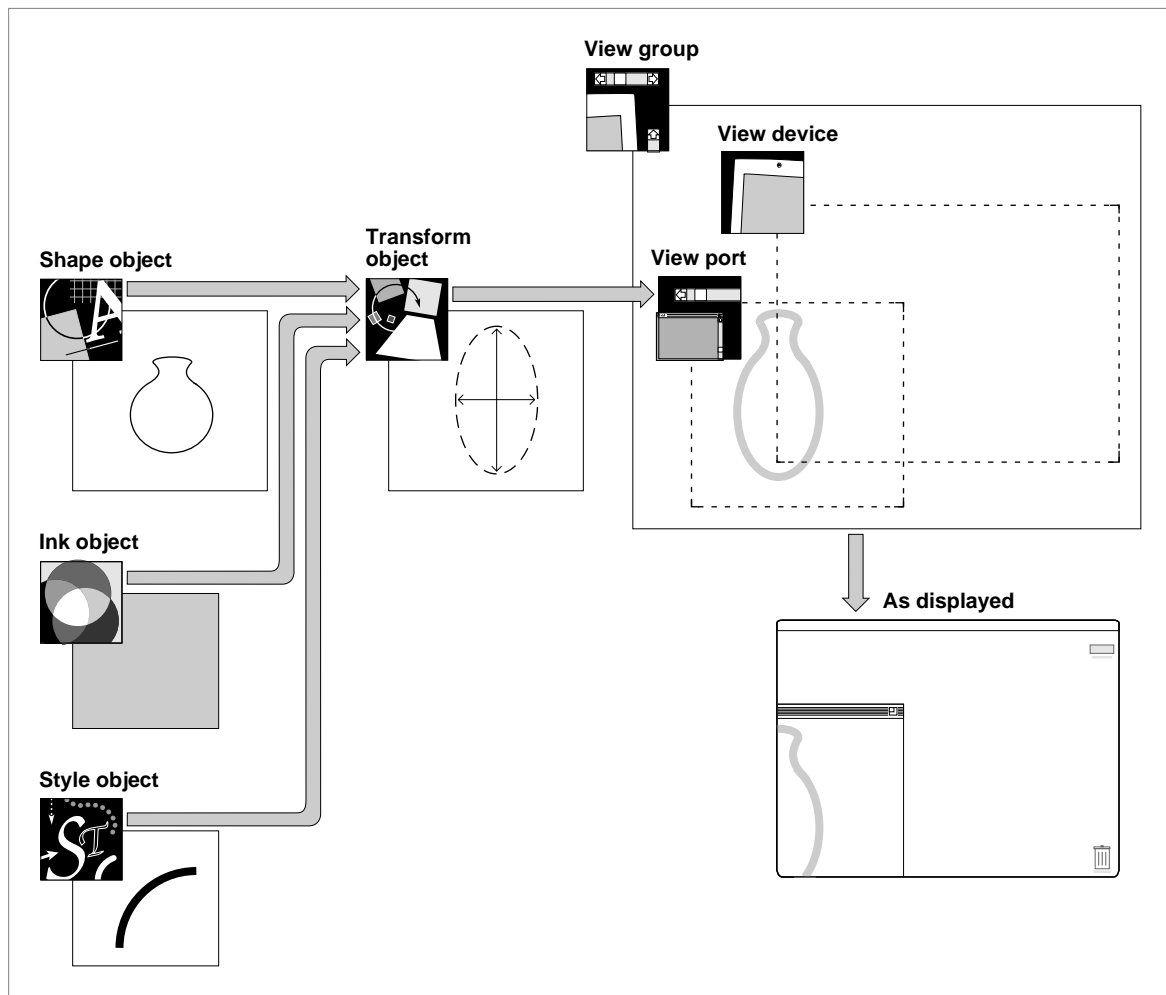
Figure 1-5 How QuickDraw GX draws a shape

Figure 1-5 is a simple case in which a single shape and its transform are drawn to a single view port that partially intersects a single view device in the same view group. Quickdraw GX provides much greater flexibility, allowing for complex combinations of shapes, transforms, view ports, view devices, and even view groups:

- Several shape objects can reference the same transform object, allowing these shapes to be scaled, rotated, and otherwise changed in unison.
- Several transform objects can reference the same view port object, allowing shapes that are transformed in different ways to appear in the same view port.

Introduction to QuickDraw GX

- A single transform object can reference several view port objects, allowing a single shape to appear simultaneously (even with different scaling or orientation) in several view ports.
- View ports can exist in a hierarchy, in which one view port “contains” another, and thus its movement, scaling, and clipping affect view ports lower in the hierarchy.
- Within a view group, view ports and view devices can overlap in any combination. Drawing occurs automatically wherever the visible portions of any view port and any view device overlap.
- More than one view group can exist simultaneously, allowing for offscreen drawing. Furthermore, the view ports referenced by the transform of a single shape need not all be in the same view groups, allowing for simultaneous onscreen and offscreen drawing of a shape.

For further discussion and illustration of these display possibilities, see the chapter “View-Related Objects” in this book.

The Drawing Sequence: Coordinate Conversion

This section discusses the sequence of events, in terms of the mappings applied to a shape, that occur in drawing. To understand the details of the transformations that take place, you must understand the coordinate spaces whose relationships are determined by the mappings contained in various objects.

The information given in this section is an abbreviated version of the discussion of mapping and clipping in the chapter “View-Related Objects” in this book. Please see that chapter for additional information, especially about the role of clipping in drawing.

QuickDraw GX Coordinates

A *coordinate space* in QuickDraw GX consists of a plane in which positions are determined by coordinates. All coordinates in QuickDraw GX are specified with fixed-point numbers in the range of -32,768.0 to approximately 32,768.0. Fixed-point numbers and the functions for manipulating them are described in the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. Coordinates are always written in the order (x, y), and for any coordinate space the point (0.0, 0.0) represents the origin of the space. Points that lie to the right of the origin increase in a positive direction along the x-axis; points that lie below the origin increase in a positive direction along the y-axis.

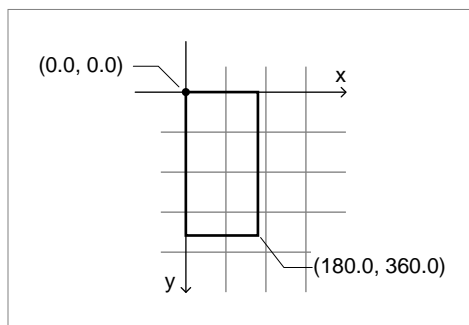
QuickDraw GX allows you to work in four coordinate spaces: geometry space, local space, global space, and device space. You can work separately in each space as appropriate; QuickDraw GX automatically converts among them when drawing. The spaces are described in order of their transformation during drawing.

Geometry Space

QuickDraw GX starts the drawing process by using the values in a shape's geometry. *Geometry space* is the space within which the fundamental position and dimensions of a shape object are defined. The numerical values in a shape's geometry define the shape's dimensions in geometry space.

Suppose, for example, that the geometry of a rectangle consists of the points (0.0, 0.0) and (180.0, 360.0), as shown in Figure 1-6. In geometry space, the rectangle's origin is at (0.0, 0.0), its height is twice its width, and its area is 64,800.0 units square. No distance metric, such as points per inch, is defined for geometry space. Thus, the absolute size of a shape is undefined in geometry space.

Figure 1-6 A rectangle in geometry space

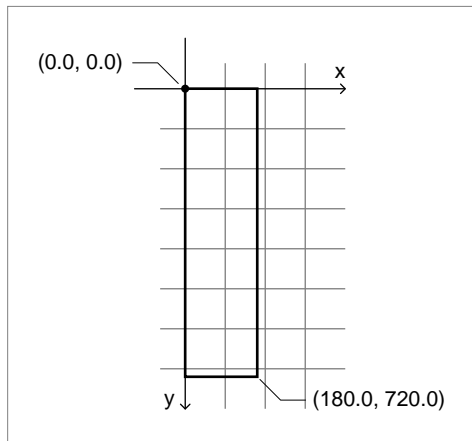


Geometry Space to Local Space

QuickDraw GX next modifies the shape's geometry by applying first the clip and then the mapping contained in the transform object attached to the shape. You typically use the transform's clip and mapping for application-specific purposes related to masking, moving, and distorting shapes within a document.

Local space defines the location and dimensions of a shape after it has been modified by the transform mapping (as well as the style properties and the transform clip). Because mappings can translate, scale, rotate, skew, and otherwise distort geometries, the dimensions of a shape in local space can be quite different from what they are in geometry space.

For example, if the rectangle shape discussed in the previous section had an associated transform whose mapping did nothing but scale the shape by 2.0 in the y-direction, its coordinates in local space would be (0.0, 0.0) and (180.0, 720.0), as shown in Figure 1-7. Its origin in local space would still be at (0.0, 0.0), but its height would be four times its width, and its area would be 129,600.0 units square. Like geometry space, local space has no distance metric. The absolute size of a shape is still undefined in local space.

Figure 1-7 A rectangle in local space (transform mapping applied)

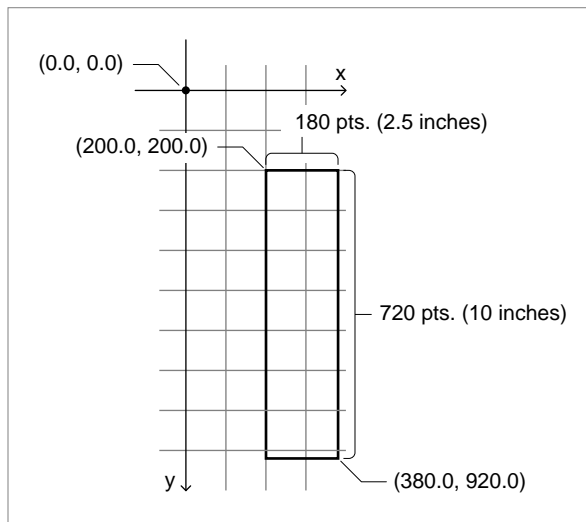
The transform object includes a reference to a view port object, and local space orients a shape within its view port. Local space is the coordinate system interior to, or local to, that view port—hence the name *local*. Thus, the rectangle example in this section would have the same local coordinates—that is, the same position and shape within its view port—no matter how the view port itself might be scaled or distorted by its own mapping when it is converted to global space (described next).

Local Space to Global Space

QuickDraw GX next modifies the shape's dimensions by applying first the mapping and then the clip contained in the view port object attached to the shape's transform. You typically use the view port's mapping to position the contents of the window you are drawing into, and you use its clip to restrict drawing to the interior of the window.

Global space defines the location and dimensions of a shape after the mapping (and clip) in its associated view port has been applied. Global space defines the real-world location and dimensions of a shape: coordinate values in global space represent distance in points (72 per inch) from the origin of the view group that the view port is part of. (Because it is the view group that relates view ports to view devices, objects in global space can have a specific spatial relationship with view devices, as described in the next section.)

For example, if the view port associated with the rectangle shape discussed in the previous sections had a mapping that did nothing but move the shape horizontally by 200.0 and vertically by 200.0, the shape's coordinates in global space would be (200.0, 200.0) and (380.0, 920.0), as shown in Figure 1-8. Its origin in global space would then be at (200.0 points, 200.0 points), its height would still be four times its width, and its area would be 129,600.0 points square (25 square inches).

Figure 1-8 A rectangle in global space (view port mapping applied)

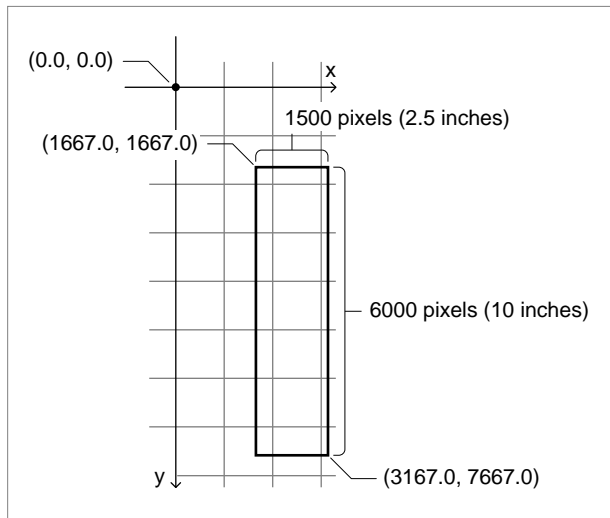
Thus, once a shape's dimensions have been converted from geometry space to local space to global space, they have a specific size and location and spatial relationship to other shapes in that view group. What remains for drawing, then, is for QuickDraw GX to convert this absolute (but device-independent) information to device-specific locations on output devices with specific pixel resolutions. That's where device space comes in.

Global Space to Device Space

Finally, QuickDraw GX modifies the shape's dimensions by applying first the mapping and then the clip of any view device object in the same view group as the view port. **Device space** defines the location and dimensions of a shape as displayed on a particular output device. The upper-left corner of the displayable area of a view device is at coordinate (0.0, 0.0) in device space. Unit distance between coordinates in device space represents one picture element, or pixel.

The view device's mapping defines both its location in global space (as a translation factor) and its pixel size (as a scaling factor). For example, if your device is a 600 dots-per-inch printer, QuickDraw GX converts global space to device space when drawing by scaling each pixel by 8.33333, which is $600/72$.

If the view device to which the rectangle shape discussed in the previous sections is drawn has a mapping that specifies no translation and a scale factor of 8.33333 both horizontally and vertically, that means that the view device's upper left corner is at (0.0, 0.0) in global space and its pixel resolution is 600 per inch. In device space, then, the dimensions of the rectangle would be (1667.0, 1667.0) and (3167.0, 7667.0), as shown in Figure 1-9.

Figure 1-9 A rectangle in device space (view device mapping applied)**Identity mapping**

A mapping that contains values such that it has no effect at all when applied to a shape is called the *identity mapping*. If the identity mapping is used for all mappings involved in drawing, a shape's geometry directly defines its absolute size and position (in points), and the shape is rendered on a view device at a resolution of 72 pixels per inch. ♦

It is seldom necessary to work in device space unless you are manipulating or hit-testing device bitmaps, because QuickDraw GX performs this kind of conversion for you. Most commonly, you define shapes in geometry space (using shape geometry), you position and modify them in local space (using the transform mapping), and you position and scale their view ports in global space (using the view port mapping).

Hit-Testing

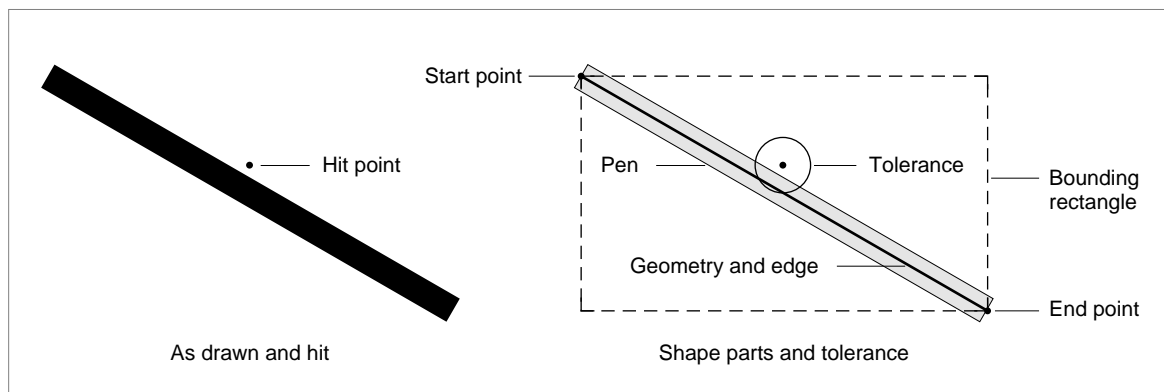
Hit-testing is the process of converting a point in the displayed representation of a shape to a location in the shape object's geometry. For example, when the user clicks the mouse button, hit-testing can tell you what displayed shape, and which part of that shape, the cursor was close to at the moment of clicking. You use hit-testing to select shapes or specific parts of shapes for highlighting or user manipulation, or to position the caret in text and to highlight text ranges. In a sense, hit-testing is the opposite of drawing, because it is a conversion from display representation to internal representation.

When you hit-test a shape, QuickDraw GX generally allows you to determine which part of a shape's geometry corresponds (within a certain tolerance) to the point you are testing against. *Tolerance* is the distance from a shape or shape part that a hit point can be and still be considered a successful hit. QuickDraw GX provides the following hit-testing functions:

- `GXHitTestShape` tests a point in local space against a shape's geometry.
- `GXHitTestPicture` tests a point in local space against a picture shape.
- `GXHitTestLayout` tests a point in local space against the text of a layout shape.
Note that you can also use `GXHitTestShape` to test layout shapes, but the kind of information it returns is different from what `GXHitTestLayout` returns.
- `GXHitTestDevice` tests a pixel (a point in device space) against a shape's geometry.

When you use a hit-testing function that returns a shape part, such as `GXHitTestShape`, the parts of a shape's geometry that you can hit-test for depend on the kind of shape. For example, for a typographic shape, the possible parts could be the bounding box, left side, right side, or side bearing of a glyph. For a line, the possible parts include its bounding rectangle, its geometry, its pen area, and its edges. Figure 1-10 shows the parts of a line involved in a particular hit-test. Shape parts are described in more detail in the chapter “Transform Objects” in this book.

Figure 1-10 Parts of a line for hit-testing



When you set up a hit-test using `GXHitTestShape`, you specify a tolerance and you also specify which parts of the shape to test against. The `GXHitTestShape` function returns all specified parts that are within the distance of the hit point defined by the tolerance. For example, if the hit point in Figure 1-10 is less than the tolerance away from the geometry part, the function could determine that the hit point corresponds to the bounds part, the geometry part, the pen part, and the edge part, depending on which of those shape parts you specify in the test.

The `GXHitTestShape` function analyzes shape parts in a specific order, and returns the distance from the hit point to the first part it encounters that is considered a hit. If you want to know the distance the hit point is from the pen, for example, you need to exclude both the bounds and the geometry parts from the test, because `GXHitTestShape` tests those first.

Introduction to QuickDraw GX

The `GXHitTestShape` function is described in the chapter “Shape Objects” in this book. The `GXHitTestPicture` function is described in the picture shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. The `GXHitTestLayout` function is described in the layout carets, highlighting, and hit-testing chapter of *Inside Macintosh: QuickDraw GX Typography*. The `GXHitTestDevice` function is described in the chapter “View-Related Objects” in this book.

Printing With QuickDraw GX

From the point of view of your application, printing with QuickDraw GX is not fundamentally different from other types of drawing. The functions you use for drawing to the screen are the same functions you use for sending images to a printer. The printing component of QuickDraw GX allows you to draw shape objects and to use the information in other objects (such as style, ink, transform, and color set) in the same way you do when drawing to the screen. When printing, the printer is represented by view port and view device objects, just as in other drawing.

To control these printing capabilities, your application creates printing-related QuickDraw GX objects before it prints a document for the first time. Your application flattens and stores those objects when it saves the document, and it retrieves and unflattens those objects when it reopens the document. The objects include the job object (the primary holder of printing information), the format object (specifying scaling and page dimensions), and the paper-type object (specifying a paper-type name and dimensions). These objects also include references to *collection objects*, which are similar to QuickDraw GX objects but are managed by the Collection Manager. The Collection Manager is described in the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

QuickDraw GX prepares an document for printing by *spooling* it, which means flattening its shapes and storing them along with the associated printing objects as a *print file*. To actually print the document, QuickDraw GX *despools* the print file and sends its data to the printer. QuickDraw GX can also use the printing process create a *portable digital document (PDD)*, which is a kind of print file that contains sufficient object and font information that it can be displayed or printed on any QuickDraw GX system, regardless of what fonts or printers are installed.

QuickDraw GX printing is based on a message-passing architecture. For example, QuickDraw GX sends a message when it wants to print a page, display a dialog box on the user’s screen, or initialize a job object. Therefore, in addition to manipulating printing objects and collection objects, your application needs to be able to respond to QuickDraw GX messages for some basic printing actions, such as updating windows behind dialog boxes.

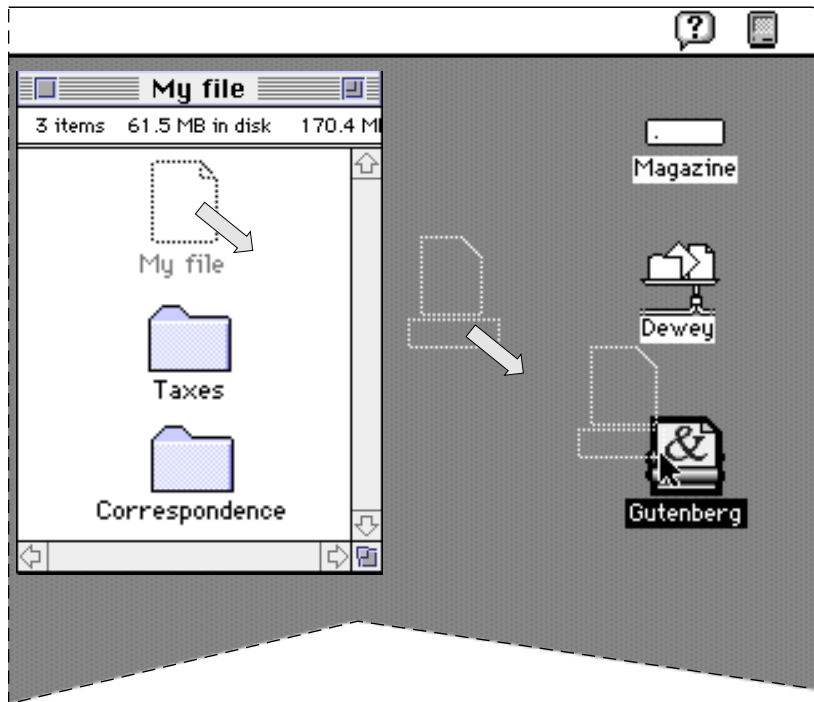
Printing extensions and printer drivers also use printing messages. A *printing extension* is an add-on software module that allows you to extend the printing functionality provided by applications and printer drivers. A *printer driver* controls how the contents of a document are spooled, rendered, and sent to a specific output device. The messaging technology used with QuickDraw GX is described in the Message Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. How printing extensions and printer drivers use printing messages, and information on how to write an extension or driver, are described in *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

The rest of this section summarizes the QuickDraw GX printing features of most interest to application developers. For more information on printing than is provided here, see *Inside Macintosh: QuickDraw GX Printing*.

Core Printing Features

QuickDraw GX provides several core features you can use to implement basic printing capabilities:

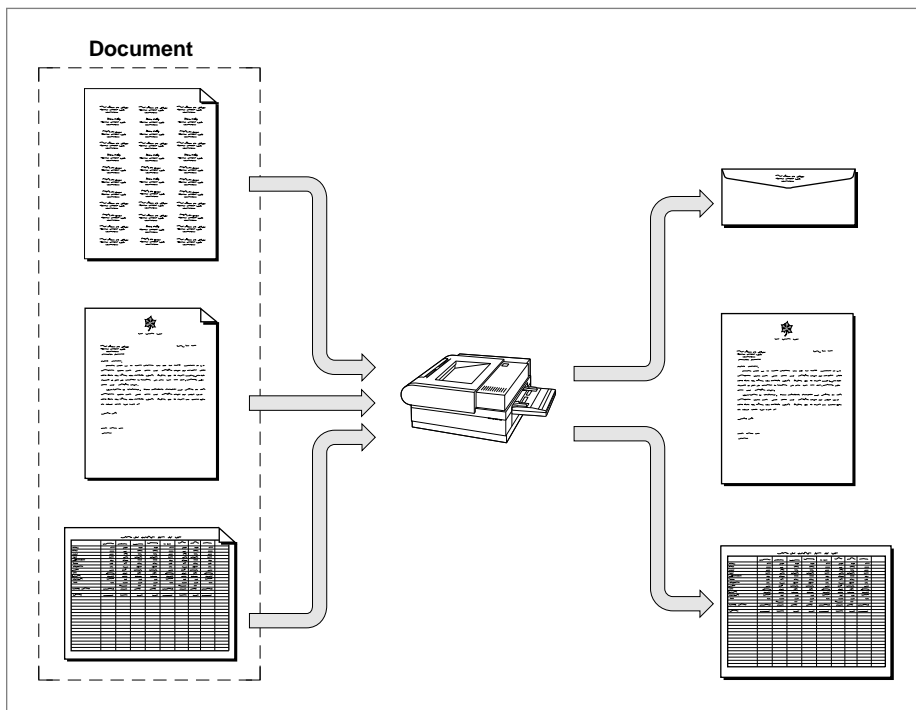
- You can create and manipulate common QuickDraw GX printing objects. For example, you create a job object for every printable document, and that job object references two printer objects: a formatting printer and an output printer. QuickDraw GX allows a user to specify a formatting printer that is different from the output printer, so that QuickDraw GX can consistently format to the device used for final output, while permitting drafts to be printed on a different printer.
- You can print documents in either of two ways. If your application stores each page as a single picture shape, you can print a page at a time with a single command. Otherwise, you can print each page by drawing, in turn, all the shapes that make up that page. QuickDraw GX captures those drawing commands and sends the images to the printer.
- You can display QuickDraw GX printing dialog boxes. You use QuickDraw GX functions to display these expandable, movable modal dialog boxes that allow users to view windows that would otherwise be obscured, and you override a QuickDraw GX printing message to permit updating of windows behind the dialog box as it is moved. For general information on movable modal dialog boxes, see the Dialog Manager chapter of *Inside Macintosh: Macintosh Toolbox Essentials*.
- You can support printing to desktop printers. A *desktop printer* is represented by an icon on the user's desktop. To print a document to a desktop printer, a user drags a document to the desktop printer icon, or else selects it and chooses the Print command from the Finder's File menu. A user can create multiple desktop printers. Figure 1-11 shows the document "My File" being printed to the desktop printer "Gutenberg."

Figure 1-11 Dragging a document to a desktop printer icon on the desktop

Custom Dialog Boxes and Page Formats

QuickDraw GX allows you to customize some of its printing features to address the needs of your particular application:

- You can add panels to QuickDraw GX dialog boxes, to provide special features that require additional user specification. For example, your application can add a panel that provides special color options for the user to select, such as color separation and color choices.
- You can manipulate the objects that handle page formatting, allowing users to specify unique formats for individual pages of a printable document. For example, your application can allow a user to create and print a single document that consists of an address page on an envelope, a business letter on a page in portrait orientation, and a spreadsheet on a page in landscape orientation. See Figure 1-12 for an example of this.

Figure 1-12 Printing a single document that has multiple formats

Advanced Printing Features

QuickDraw GX includes several advanced printing features that allow your application to provide additional capabilities for users and to optimize output for particular printers:

- You can use direct mode printing, which takes advantage of a printer's built-in features, such as fast text streaming with built-in fonts.
- You can use alternative representations of QuickDraw GX objects. When printing, QuickDraw GX translates the objects of a shape into device-specific information. For optimum performance on particular devices, you can assign specialized tag objects known as *synonyms* to printed shapes and associated objects, to provide an alternative representation of the graphics objects. You can also use tag objects to select specific printing options, such as pen table information, for vector devices.
- You can display your own printing status information. QuickDraw GX allows you to prevent the display of the standard QuickDraw GX Status dialog box during printing and to substitute status information from your own application.
- You can open and display the pages of a PDD or other print file. If QuickDraw GX is installed, any application—including applications that are not QuickDraw GX-aware—can create a document that can be viewed or printed from any other computer that has QuickDraw GX installed. The PDD also provides font security in that the font data is “locked” into the document and only the minimum font information is contained therein.

The QuickDraw GX Programming Environment

QuickDraw GX is more than a framework for creating and manipulating objects; it is also a programming environment with many features designed to aid application development. This section describes some of these features and some ways to approach programming with QuickDraw GX.

Setting Up QuickDraw GX Memory

Your application enters the QuickDraw GX environment by creating a graphics client. A *graphics client* is an object that represents a memory environment set up for your application by QuickDraw GX. It consists of a QuickDraw GX heap and the global variables needed by QuickDraw GX. It represents your application's individual QuickDraw GX world.

Normally, each application creates and uses a single graphics client, although it is possible to create and use more than one at a time. In most cases, you don't even explicitly set up a graphics client at all; one is created for you as you begin making QuickDraw GX calls to create and use objects. For more information, see the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Handling Errors

In all but its printing component, QuickDraw GX uses a sophisticated, three-level system for reporting diagnostic messages. The execution of a function may result in the generation of an *error*, a *warning*, or a *notice*:

- Errors represent the most severe problems, and occur when a function is unable to execute.
- Warnings occur when a function has completed but may have provided an incorrect or unexpected result.
- Notices occur when unnecessary or redundant actions have been performed. (Notices are available only in the debugging version of QuickDraw GX; see "Debugging and Non-Debugging Versions" on page 1-39 for more information.)

Errors, warnings, and notices are not returned as function results. Instead, they are *posted*, or stored by QuickDraw GX in locations accessible through function calls. To determine whether, for example, an error has occurred, your application makes a specific call (such as `GXGetGraphicsError`) that returns not only the most recent error but also the first error posted since the last time you called `GXGetGraphicsError`. For information about these function calls, see the debugging chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Introduction to QuickDraw GX

You can use the error-handling facilities of QuickDraw GX in the following ways:

- You can install an error-handling function that QuickDraw GX calls whenever an error, warning, or notice occurs.
- You can post errors, warnings, or notices yourself from your own application's functions.
- You can tell QuickDraw GX to ignore specific warnings or notices. You can create and manipulate a list of warnings and a list of notices to be ignored.

Functions within the printing component of QuickDraw GX do not use this system for reporting diagnostics. Instead, most functions place errors directly into the job object involved. Some printing functions return a function result of type `OSErr`, that describes a Macintosh error code. For more information, see the core printing features chapter of *Inside Macintosh: QuickDraw GX Printing*.

Debugging

QuickDraw GX provides both a debugging and non-debugging version of the software. In addition, QuickDraw GX provides a low-level debugger, similar to MacsBug, that allows you to examine internal data structures. This section summarizes these approaches to debugging. For more information, see the debugging chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Debugging and Non-Debugging Versions

There are two versions of QuickDraw GX. The debugging version is intended for application development and is meant for use by software developers only. The non-debugging version is intended for running completed applications and is the publicly released version of QuickDraw GX.

The debugging version of QuickDraw GX provides extensive error handling. It posts all three levels of diagnostic messages (errors, warnings, and notices), and it provides special functions to assist in the posting, utilization, and control of debugging messages. The debugging version allows you to perform validation checking on both QuickDraw GX objects and your own application parameters at each function call. The debugging version also includes the `GXGetShapeDrawError` function, which can give you very specific information on why a particular shape may not have drawn correctly.

The non-debugging version of QuickDraw GX has much less extensive error handling. It reports only two levels of result messages (errors and warnings), and only a limited number of them. In the non-debugging version, errors and warnings are mostly limited to out-of-memory and range-checking messages.

Debugging With GraphicsBug

GraphicsBug is a tool you can use to track down bugs in a QuickDraw GX application. Its mode of use and its command set are analogous to MacsBug. GraphicsBug works with both the debugging and non-debugging versions of QuickDraw GX.

You can use GraphicsBug to check the contents of QuickDraw GX memory and to display and validate objects within memory. GraphicsBug does not allow you to create, modify, or dispose of objects. Listing 1-1 shows a sample dump of the QuickDraw GX heap created with GraphicsBug.

Listing 1-1 Sample GraphicsBug heap dump (HD) listing

Start	Length	Δ	Typ	Busy	Mstr	Ptr	Temp	TBsy	Disk	Object
00469728	0000010c+00		d		00000000			b		heap header block
00469834	0000003c+00		d		00000000					freeFileList
00469870	0000005c+00		i		00470e68					text
004698cc	00000042+02		i		00470e64					text
00469910	000000a0+00		i		00470e60					style
004699b0	00000036+02		i		00470e5c					ink
004699e8	00000060+00		i		00470e58					transform
00469a48	000000c0+00		d		00000000					port
00469b08	00000038+00		i		00470e54					full
00469b40	00007228		f		00000000					free block
00470d68	00000110+00		d		00469728			b		master pointer block
00470e78	0000000c+00		d		00469728			b		heap trailer block

	Total Blocks			Total of Block Sizes		
Free	0001	#	1	00007228	#	29224
Direct	0002	#	2	00000318	#	792
Indirect	0006	#	6	00000210	#	528
Sub Heaps	0000	#	0	00000000	#	0
Heap Size	0009	#	9	0000775c	#	30556

The listing shows the objects that you create as well as private QuickDraw GX objects. From the heap dump, you can look into the contents of these objects using additional GraphicsBug commands. For a complete list of commands, type ? on the GraphicsBug command line.

Note

Do not use GraphicsBug to make assumptions about the structure of objects in memory; object structure is subject to change. ♦

For examples of the use of GraphicsBug in analyzing flattened shapes, see the stream format chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Programming Conventions and Consistencies

The QuickDraw GX programming environment provides many consistent features and conventions to make graphics software development more convenient and more efficient. This section lists some of them.

Object Behavior

Many QuickDraw GX objects have similar features and consistent behavior, in ways such as the following:

- In general, setting a property of an object causes action or new behavior only when needed—which may not be immediately. For example, setting the `gxDiskShape` attribute of a shape object, which instructs QuickDraw GX to write the shape to disk, takes effect only as soon as QuickDraw GX needs memory space and looks for objects to unload.
- QuickDraw GX handles object properties consistently; it does not change a property once you have set it. For example, if you set an object reference to `nil` in order to use the default version of an object, QuickDraw GX does not replace that `nil` value with an actual reference. If you later make another call to retrieve that reference, you will get back the `nil` value you originally set. (A minor exception to this rule occurs with certain calls that assign arrays to the style object associated with typographic shapes; QuickDraw GX may alter the order of elements in those arrays. See the layout styles chapter in *Inside Macintosh: QuickDraw GX Typography* for more information.)
- Most objects can be explicitly moved into and out of memory, using `GXLoadObject` and `GXUnloadObject` functions. View-related objects and printing objects are exceptions to this rule.
- Many objects can have tag objects attached to them, using `GXGetObjectTags` and `GXSetObjectTags` functions. View group objects, printing objects, font objects, and tag objects themselves are exceptions to this rule.
- Most objects can be shared, and thus have `GXCloneObject` and `GXGetObjectOwners` functions. View-related objects are exceptions to this rule; they are shared but they have no owner count and cannot be cloned.

Functions and Function Results

QuickDraw GX functions are designed to operate in a consistent manner, as follows:

- Most QuickDraw GX functions do not return error codes as function results. Instead, they return object references or pointers to structures. This makes nesting of calls easier.
- Functions are consistently named and have consistent behavior across all objects. Most objects have similarly behaving `GXNewObject`, `GXDisposeObject`, `GXCopyToObject`, and `GXEqualObject` functions. The property-accessing `GXGetObjectProperty` and `GXSetObjectProperty` functions behave consistently, using index values and ranges for inserting, deleting, or replacing all or parts of arrays.
- All functions, except some printing functions that return an `OSErr` value, return `nil` or zero as a function result if an error occurs.

Introduction to QuickDraw GX

- If a function posts an error, it does not modify any data or objects that are input parameters to the function.
- Functions of the form *GXGetObjectProperty* that fill out an array that is passed as a parameter typically return the number of elements in that array as a function result. For example, the *GXGetTransformViewPorts* function, used to get the list of view port references in a transform object, returns the number of elements in the list as its function result. Thus, you commonly call such a function twice in a row: first to determine the size of array to allocate, and second to obtain the filled-out array itself.
- Many functions of the form *GXSetObjectProperty*, which modify a property of a particular object, have a parallel function of the form *GXSetShapeProperty*, which performs the same function but allows you to specify instead the shape object that references the affected object. If the object whose property is changed is not shared by more than one shape, the two functions have an identical effect. If, however the object is shared, *GXSetShapeProperty* makes a copy of the object before modifying it, so that the other shapes using it are not affected unintentionally.
For example, the *GXSetTransformViewPorts* function assigns a list of view port references to the specified transform object, and the *GXSetShapeViewPorts* function assigns a list of view port references to the transform object associated with the specified shape. If the transform object is used by more than one shape, *GXSetTransformViewPorts* has the effect of altering all shapes that use that transform; *GXSetShapeViewPorts*, however, first makes a copy of the transform and then alters it, so that other shapes are not affected.
- When an out-of-memory condition occurs, it is rarely fatal. QuickDraw GX initially posts an *out_of_memory* error, but execution continues and subsequent attempts to reference the object responsible for the error result in posting of *object_is_nil* errors.
- The debugging version of QuickDraw GX provides extensive error-checking and validation capabilities to help you determine why a function call has failed.

Function Parameters

When passing parameters to a QuickDraw GX function, you can take advantage of the following design consistencies and conveniences:

- The first parameter in any function call is the object or structure acted upon.
- Parameters whose names contain the word “source” are never modified by a function; parameters whose names contain the word “target” may be modified.
- Whenever an array or structure is passed as a parameter to a function, the application is responsible for allocating it. QuickDraw GX fills out structures, but it does not allocate them.
- When a variable-sized array or structure is passed as a parameter to a function, it is preceded in the parameter list by a size or count parameter.

Introduction to QuickDraw GX

- In the C language definitions for QuickDraw GX functions, the term `const` preceding a parameter that is an array, structure, or pointer indicates that QuickDraw GX reads from, but does not write to, the data pointed to by the parameter.
- In general, passing `nil` or zero for a parameter instructs QuickDraw GX to use its default or most appropriate behavior for that situation. Thus you need to explicitly set parameters only when you need specific, non-default behavior. To actually assign a `nil` value to a property, pass the constant `gxSetToNil` (see Table 1-1).
- In functions that use coordinates, the x-coordinate (horizontal axis) is specified before the y-coordinate (vertical axis).
- For convenience in handling arrays and pointers in parameters, QuickDraw GX provides several predefined constants, as listed in Table 1-1.

Table 1-1 Convenience constants for parameters

Constant	Value	Explanation
<code>gxSelectToEnd</code>	<code>-1</code>	Used in a size or count parameter, to mean “from the current position in the array to the end of the array.”
<code>gxSetToNil</code>	<code>(void *)(-1)</code>	Used in a parameter (where a function takes more than one parameter) to assign a <code>nil</code> value to a pointer or reference property (simply passing <code>nil</code> has no effect on the property).
<code>gxNoAttributes</code>	<code>0</code>	Used to clear the attributes property of an object.
<code>gxColorValue1</code>	<code>0xFFFF</code>	Used to specify the maximum value for a color-component, which is interpreted to mean 1.0
<code>gxAnyNumber</code>	<code>1</code>	Used as an index in an array declaration to indicate that the array is not of any specific size.

Implementation limits

Limits on valid parameter values or on the sizes of structures or behaviors of objects may depend on the current implementation of QuickDraw GX, and may be different from the fundamental limits imposed by the programmatic interface itself. For example, a parameter to a function may be a long, but the range of acceptable values for that parameter may be much smaller than the full range of values that can fit into a long. ♦

Code Naming Conventions

QuickDraw GX uses these naming conventions to provide consistency across the application interface:

- Function names begin with uppercase GX—for example, `GXDrawShape`. Important exceptions are those in the Collection Manager and those that are mathematical functions because those functions can be useful outside of the QuickDraw GX environment.
- Identifiers of constants and data types defined by QuickDraw GX begin with lowercase `gx`—for example, `gxWindingFill` and `gxShapeType`. One exception is the type `Fixed`, which represents a QuickDraw GX fixed-point number but does not have a `gx` prefix. Types defined by the programming language itself, such as `short`, do not have a `gx` prefix.
- Names of fields in data structures, and parameter names in function prototypes, begin with lowercase letters and do not have a `gx` prefix.
- An enumeration that defines several constants is usually named with a plural form—for example, `gxDashAttributes`. Such an enumeration is commonly paired with a type definition that is a singular form of the same name—for example, `gxDashAttribute`. You can use the type to specify one of the enumerated values for a parameter or field.
- Object attributes have suffixes that identify the kind of object they apply to. For example, dash attributes specified by the `gxDashAttributes` enumeration include the attributes `gxBendDash` and `gxAutoAdvanceDash`.

Relationship to the Macintosh Toolbox

QuickDraw GX is in general designed to be platform independent. Within the QuickDraw GX environment, the programming interface does not depend on the existence of the Macintosh Toolbox or Macintosh hardware.

However, when running on a Macintosh computer, QuickDraw GX still must have an interface with the Macintosh Toolbox. QuickDraw GX does not create windows, handle menus, receive keystrokes or automatically track mouse movements (although it does support hit-testing). Therefore, for basic input and output needs, QuickDraw GX includes several sets of functions that carry information from the QuickDraw GX environment to the Macintosh world and back:

- You can associate QuickDraw GX view ports with Macintosh windows, which restricts a view port to the current size of the window and prevents drawing outside the content area of the window. QuickDraw GX and the Macintosh Window Manager then manage the view port for you such that, if the user moves the window, the view port moves too, or if the user changes the size of a window, the drawable area in the view port also changes.
- You can associate a QuickDraw GX view device with a Macintosh graphics device (`GDevice`).

- You can translate coordinate locations, including mouse locations, between the integer-based QuickDraw global space and the fixed-point QuickDraw GX coordinate spaces.
- You can convert QuickDraw calls to QuickDraw GX calls, in two ways. You can use one set of functions to set up a situation whereby all QuickDraw calls are captured and converted to QuickDraw GX shapes in a QuickDraw GX picture. You can use another function to directly translate QuickDraw pictures to QuickDraw GX pictures.

See the Macintosh environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities* for information about these functions.

Summary Table and Diagram of QuickDraw GX Objects

QuickDraw GX provides at least 17 objects that you can manipulate. Table 1-2 lists these objects and summarizes their characteristics. Following Table 1-2, Figure 1-13 on page 1-49 diagrams the relationships among the basic QuickDraw GX objects, and shows the object properties of each.

Table 1-2 QuickDraw GX objects

Object	Description
Basic QuickDraw GX objects	
Shape	Defines the basic representation of a drawable entity. A shape object describes a geometry of a certain type (such as a line, rectangle, bitmap, or text) and how the geometry is framed or filled when drawn. A shape also has references to its three related objects: style, ink, and transform. See the chapter “Shape Objects” of this book for more information. Graphic shape types are described in <i>Inside Macintosh: QuickDraw GX Graphics</i> ; typographic shape types are described in <i>Inside Macintosh: QuickDraw GX Typography</i> .
Style	Describes certain characteristics affecting how a shape is drawn. For geometric shapes, this includes the thickness of the pen, the starting and ending caps for line segments, joins between line segments, and the dash or pattern to be applied to the shape. For typographic shapes, it includes the font, text size, and typeface of the text. See the chapter “Style Objects” in this book, the geometric styles chapter of <i>Inside Macintosh: QuickDraw GX Graphics</i> , and the typographic styles and layout styles chapters of <i>Inside Macintosh: QuickDraw GX Typography</i> for more information.

continued

Table 1-2 QuickDraw GX objects (continued)

Object	Description
Ink	Describes a shape's color and its transfer mode (how the color is applied when the shape is drawn). Ink objects support many different kinds of color specification, and many different transfer modes. An ink object can reference a color set object or color profile object or both. See the chapters "Ink Objects" and "Color-Related Objects" in this book for more information.
Transform	Describes the clip and mapping applied to a shape when it is drawn. The clip limits the extent of the shape when it is drawn; it may be described by any primitive shape geometry (except picture, text, layout, and multi-bit bitmap). The mapping defines translation, scaling, skewing, rotation or perspective. The transform object also describes the criteria used for hit-testing the shape. Transforms have references to one or more view port objects. See the chapter "Transform Objects" in this book for more information.
Color set	Contains an indexed set of colors; analogous to a color table. Color sets are used when colors are specified by index instead of by direct color value. Bitmaps commonly use color sets. See the chapter "Colors and Color-Related Objects" in this book for more information.
Color profile	Contains color matching information. The information in a color profile can be used to convert device-specific colors to device-independent colors and back. To provide the most faithful reproduction of colors on different devices, QuickDraw GX automatically performs color matching with available color profiles whenever it draws. See the chapter "Colors and Color-Related Objects" in this book for more information.
View port	Defines the location into which a shape is drawn. A view port object describes the clip and mapping associated with a window (or a part of a window, such as a pane). The mapping defines the location, scale, and orientation of the view port in QuickDraw GX global coordinates. A view port specifies the dithering or halftones used by every object that draws into this window. View ports can be arranged in a hierarchy. See the chapter "View-Related Objects" in this book for more information.
View device	Describes the clip, mapping, and bitmap associated with a physical display device such as a monitor or printer. The mapping describes the view device's position and resolution in QuickDraw GX global coordinates. The bitmap defines the dimensions of the device, the number of bits per pixel, the color representation of each pixel value, and the color profile. See the chapter "View-Related Objects" in this book for more information.

Table 1-2 QuickDraw GX objects (continued)

Object	Description
View group	Describes an imaging world that is the global space in which view ports and view devices are located. Within a view group, view ports and view devices can overlap each other in any combination; the intersection of each view port with a view device determines what is actually visible on that device. Multiple view groups allow for offscreen drawing, in which view ports or view devices can have the same positions without interfering with each other, since they are in different coordinate spaces. See the chapter “View-Related Objects” in this book for more information.
Tag	Contains any kind of information an application wants to add to a QuickDraw GX object. Tag objects are general containers that can have anything in them, from labels to alternate drawing instructions to anything else you feel is useful. You can attach a tag object to the tag list of most kinds of objects (except tag objects themselves). See the chapter “Tag Objects” in this book for more information.
Printing objects	
Job	Holds the primary printing information for a document. Every printable document has a job object associated with it. The job object specifies a number of copies and a page range, and includes references to one or more format objects and two printer objects. See the core printing features chapter of <i>Inside Macintosh: QuickDraw GX Printing</i> for more information.
Format	Specifies page-formatting characteristics such as scaling and page dimensions, and includes a reference to a paper-type object. See the core printing features chapter of <i>Inside Macintosh: QuickDraw GX Printing</i> for more information.
Paper type	Specifies a paper-type name (such as “US Letter”), the physical dimensions of the paper, and the printable area within it. See the core printing features chapter of <i>Inside Macintosh: QuickDraw GX Printing</i> for more information.
Printer	Represents the capabilities of a physical printer and includes a name and type, a driver name and type, and a reference to one or more view device objects that represent imaging areas, and from which you can retrieve information. See the advanced printing features chapter of <i>Inside Macintosh: QuickDraw GX Printing</i> for more information.
Print file	Represents the file that results from spooling, which is the preparation of a printable representation of a document. See the advanced printing features chapter of <i>Inside Macintosh: QuickDraw GX Printing</i> for more information.

continued

Introduction to QuickDraw GX

Table 1-2 QuickDraw GX objects (continued)

Object	Description
Other objects	
Font	Represents an available font. A font object contains information about the font's names, encodings, font variations, and other tables. See the fonts chapter of <i>Inside Macintosh: QuickDraw GX Typography</i> for more information.
Graphics client	Represents the QuickDraw GX memory allocated for an application, which is separate from the memory the application itself occupies and allocates. Each QuickDraw GX application is represented by a graphics client object. A graphics client has no accessible properties. See the memory management chapter of <i>Inside Macintosh: QuickDraw GX Environment and Utilities</i> for more information.
Collection	Contains any type of data in any structure. Used by printing objects to hold additional information such as halftoning specifications. Collection objects are not QuickDraw GX objects; they are managed by the Collection Manager. See the Collection Manager chapter of <i>Inside Macintosh: QuickDraw GX Environment and Utilities</i> for more information.

The following figure, Figure 1-13, shows the relationships among the basic QuickDraw GX objects and lists the properties of each object. The appropriate portion of this figure is reproduced in each chapter that describes a specific kind of object.

Note that, in Figure 1-13, properties that are references (or arrays of references) to other objects are shown in italics; for most of those properties, an arrow extends to the diagram of the referenced object. For clarity, however, some of the arrows are not shown. For example, no object's tag list has an arrow pointing to the diagram of the tag object. For the same reason, the properties of the view port that reference other view ports have no attached arrows.

For a diagram showing all the properties of the printing objects, see the introductory chapter of *Inside Macintosh: QuickDraw GX Printing*. For a diagram showing the contents of the geometry of each type of shape object, see the chapter "Shape Objects" in this book.

Figure 1-13 Properties of the basic QuickDraw GX objects

