

Page Formatting and Dialog Box Customization

This chapter describes how your application can manipulate the objects that QuickDraw GX uses to format the pages of a document or add panels to QuickDraw GX dialog boxes.

Read the information in this chapter if you want your application to allow users to specify unique formats for the individual pages of a printable document. For example, using QuickDraw GX, 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 sheet of paper in portrait orientation, and a spreadsheet on a sheet of paper in landscape orientation.

You should also read this chapter if you want to add panels to QuickDraw GX print dialog boxes. For example, your application may add a panel that allows the user to specify additional information, such as color-separation for color printing.

Before you begin using QuickDraw GX page formatting and dialog box customization features, you should be familiar with the basic concepts for printing with QuickDraw GX, as described in the chapter, “Introduction to Printing With QuickDraw GX.” You should also be familiar with creating and manipulating a job object, as described in the chapter “Core Printing Features” in this book.

This chapter begins by summarizing what you need to know to support the page formatting and dialog box customization features of QuickDraw GX. Because page formatting and dialog box customization can use collection objects, this topic is introduced first. Page formatting is discussed next because you can use collection items as parameters to specify formatting criteria. Dialog box customization is discussed after the other two topics because you may need to use nondefault dialog boxes to allow the user to set the values of items in a collection object. Keep in mind that any QuickDraw GX print dialog box can be customized, not just the Custom Page Setup dialog box associated with page formatting.

After introducing the basic concepts associated with printing-related collection objects, page formatting, and dialog box customization, this chapter shows you how to

- access an item in a collection object for use with a dialog box
- keep track of format objects that are shared by multiple pages of a document
- create a format object for a page in a document
- clone a format object for multiple pages in a document
- dispose of a format object for a page in a document
- access information associated with a format object
- display the Custom Page Setup dialog box
- support special formatting features
- associate format objects with document pages
- add panels to QuickDraw GX dialog boxes
- automate panel information

About Page Formatting and Dialog Box Customization

Page formatting is the ability to format individual pages of a document differently from the default format for the document. The available formats are specified by the printer driver. You specify a format object when you print each page of a document. If you specify the first format in the job object's format list, the default format for the document is used. If you specify another format, it is used to format the page. For more information about printing pages and specifying formats, see the chapter "Core Printing Features" in this book.

Typically, you associate the default format object with each page in the document and let the user choose the pages to format differently from the default. The user can choose the format with the Custom Page Setup menu item of the File menu, which displays the Custom Page Setup dialog box on the user's screen. You are responsible for associating the chosen format with the page. Thus, you need to determine which format objects are in use and save them with the job object when the document is saved. You also need to retrieve them along with the job object when the document is opened. For more information about saving and retrieving these job objects, see "Associating Format Objects With Document Pages" on page 3-61.

The Custom Page Setup dialog box provided by QuickDraw GX allows the user to format a page, remove the format and revert to the default format, change the paper type for the page, and change the page's scale and orientation. You can allow more choices by customizing this dialog box. For example, you can allow the user to specify a halftone to be applied to the page. Because the Custom Page Setup dialog box provided by QuickDraw GX does not provide an option for specifying a page halftone, the printer driver or a printing extension must customize the dialog box, or you must customize the dialog box in the application.

If you customize a dialog box, you typically gather additional information from the user, although you can also customize a dialog box to restrict the user's choices. The additional information is stored in a collection object. In the halftone example, the printer driver stores the possible halftone options in the format collection. You can customize the Page Setup dialog box to allow a halftone to be chosen for the default format, or you can customize the Custom Page Setup dialog box to allow a halftone to be chosen for a particular page.

QuickDraw GX allows you to customize any of the print dialog boxes:

- The Print dialog box, in which the user specifies parameters for printing the job.
- The Page Setup dialog box, in which the user specifies default formatting by selecting the formatting printer. This dialog box is also used to specify the default paper type.
- The Custom Page Setup dialog box, in which the user specifies formatting for a particular page, including the paper type.
- The Printing Status dialog box, in which the status of the spooling operation is displayed. This dialog box is not usually customized. You may choose, however, to suppress the display of the dialog box under certain conditions.

About Collection Objects

QuickDraw GX supports collection objects to store and to allow your application to store printing-related, formatting, and paper-type information associated with a printable document. Essentially, these collections specify additional information that are not absolutely required to print a job, format a document, or specify the kind of paper. In QuickDraw GX printing, collection objects typically store information you can use to customize dialog boxes. You can access information required by your application from these collection objects, however, whether or not you allow the user a choice in a dialog box. You can also use collection objects to store information that is of use only to your application.

You can use collection objects without customizing dialog boxes. For example, a user may print by dragging the document's icon onto a desktop printer or by choosing the Print One Copy menu item from the File menu. In these cases, your application may need to change the settings in a collection object directly, without user intervention.

You can also store information that is not already provided by QuickDraw GX. For example, as part of using QuickDraw GX page formatting features, your application is responsible for managing the correspondence between format objects and individual pages in a document. Your application can use a format collection item to store this correspondence. Storing correspondence information in a format collection is discussed in "Associating Format Objects With Document Pages," which begins on page 3-61.

Collection Tag IDs and Item IDs

When you add data (referred to as a collection item) to a collection object, the Collection Manager associates the data with a collection tag ID and a collection item tag. Together, the 4-byte collection tag ID and the 4-byte collection item tag ID uniquely identify a collection item within a particular collection object.

Note

To avoid the confusion between tag objects (which are not related to collection objects at all), collection tags, and collection item tags, this book refers to collection tags as tag IDs and to collection item tags as item IDs. Tags, when used in this book, refer to tag objects. ♦

Page Formatting and Dialog Box Customization

QuickDraw GX assigns the `gxPrintingTagID` tag ID to each of its predefined collection items:

```
enum { gxPrintingTagID = -28672 };
```

For each of its collection items, QuickDraw GX defines an item ID, such as `gxCopiesTag` for the collection item that defines the number of copies to print:

```
enum {gxCopiesTag = 'copy'};
```

QuickDraw GX reserves all tag IDs that are negative or less than 127. It also reserves all collection items defined by lowercase characters. For example, you can use your application's registered creator type for the tag ID.

In addition to the collection tag and collection item ID, the Collection Manager allows items to be accessed by index. You can use an index to provide faster access to specific items in a collection or to perform operations on all collection items in a collection object. This index does not uniquely identify an item, however, because adding or removing items can change an item's index number. For information about collection indexes and collection objects in general, see the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Item Structures

A structure defines the form of most collection items. A type definition is associated with each of these structures:

```
struct gxCopiesInfo{
    long copies;
};

typedef struct gxCopiesInfo gxCopiesInfo;
```

For example, you can use `gxCopiesInfo` as both a structure name and a data type definition:

```
gxCopiesInfo myCopies;
struct gxCopiesInfo myCopies;
```

In this book, only the structure definition is presented. Type definitions are only presented when they are not associated with a structure, as in `gxCollectionCategory`, defined in the next section.

Categories of Collection Items

If you add an item to a collection, you need to decide whether the contents will be valid if the output printer or formatting printer changes. You also must decide if the item should persist when the collection is flattened.

QuickDraw GX purges the items that are not valid after the printer driver changes, based on the contents of the collection item's user attribute bits. It also decides which items to flatten based on these bits. For general information about user attribute bits, see the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

A printer-driver switch occurs whenever a user changes the device class (type of printer) of the output printer or formatting printer associated with a particular document. For example, if the user switches output printers, QuickDraw GX discards the tray-feed information, which specifies the current paper tray, because it may have changed.

QuickDraw GX assigns collection object items into categories based on the contents of the `gxCollectionCategory` user attribute bits, as shown in the following enumeration:

```
typedef short gxCollectionCategory;

enum {
    gxNoCollectionCategory          = (gxCollectionCategory) 0x0000,
    gxOutputDriverCategory          = (gxCollectionCategory) 0x0001,
    gxFormattingDriverCategory      = (gxCollectionCategory) 0x0002,
    gxDriverVolatileCategory        = (gxCollectionCategory) 0x0004,

    gxVolatileOutputDriverCategory =
        gxOutputDriverCategory + gxDriverVolatileCategory,
    gxVolatileFormattingDriverCategory =
        gxFormattingDriverCategory + gxDriverVolatileCategory
};
```


Items in the `gxNoCollectionCategory` category are not purged. Data that is specific to an output printer driver should be grouped in the `gxVolatileOutputDriverCategory` collection item category. Data that is specific to a formatting printer driver should be grouped in the `gxVolatileFormattingDriverCategory` collection item category.

Data that need not be saved when a job is flattened should be grouped in the `gxDriverVolatileCategory` collection item category. You must also clear the `collectionPersistenceBit` attribute bit if you would like to keep the information but do not require it to be saved with the collection.

The Job Collection

QuickDraw GX primarily stores in a job collection the information contained in the Print dialog box and its General panel, Paper Match panel, and Print Time panel. Panels for the Print dialog box are discussed in the chapter “Introduction to Printing With QuickDraw GX” in this book. QuickDraw GX stores 18 items in a job collection, as shown in Figure 3-1.

Figure 3-1 The job collection



Job collection	
Print-job information	
Collation information	
Copies information	
Page-range information	
Quality information	
File-destination information	
File-location information	
File-format information	
File-fonts information	
Paper-feed information	
Manual-feed information	
Standard mapping information	
Special mapping information	
Tray-mapping information	
Print-panel information	
Format-panel information	
Paper-matching information	
Translated-document information	

A brief description of each collection item follows. To see how the pieces of data are structured in the collection item, see “Constants and Data Types for Job Collection Items” beginning on page 3-78. Job collection items include the following:

- **Print-job information.** This collection item describes the job information for the print job. It contains information such as the total number of pages to print, the print job’s priority, designated time to print, and the amount of time in which to keep a print job

in an alert state before cancelling the job. It also contains the first page from which to begin printing and indicates whether the user chose to be alerted before printing begins or after printing is finished. In addition, this property contains the name of the application used to create the printable document, the name of the user's document, and the name of the user associated with the printable document.

- **Collation information.** This collection item specifies whether document pages should be collated when printed. The user typically specifies whether collation is desired in the Collate Copies checkbox in the Print dialog box.
- **Copies information.** This collection item contains the number of copies of the document to print. The user specifies the number of copies to print in the Copies field in the Print dialog box.
- **Page-range information.** This collection item contains the page-range information in the job object as well as data that allows customized or replacement page ranges. It contains the user-specified custom, default, or replacement page-range information from the Print dialog box.
- **Quality information.** This collection item contains information about the quality mode, such as the default quality mode and the current mode. It also includes the number of quality menu items and an array of quality names (such as "Best") to display in the Quality pop-up menu in the Print dialog box.
- **File-destination information.** This collection item contains the file-destination information for the job object. It specifies whether the user chose File in the Destination pop-up menu in the Print dialog box.
- **File-location information.** This collection item contains the file-location information as a `FSSpec` structure. It typically contains the result of a call to `StandardGetFile`, which is used to determine the filename when the user prints to a file.
- **File-format information.** This collection item contains the name of the file format if the destination of the print job is a file.
- **File-fonts information.** This collection item specifies whether fonts should be stored as part of the file. If fonts are stored, it specifies whether all fonts are stored or only nonstandard fonts.
- **Paper-feed information.** This collection item contains the paper-feed information for the job object. It specifies whether the user chose the Automatic or Manual radio button for Paper Feed in the Print dialog box.
- **Manual-feed information.** This property contains the manual-feed information for the job object. It specifies the number of paper types to manually feed and an array of paper-type names to display.
- **Standard mapping information.** This collection item specifies whether to use standard mapping information for the print job. The item contains a Boolean value that is `true` if input tray paper matching is to be used.
- **Special mapping information.** This collection item contains the special mapping information for the job object. It specifies mapping options, such as whether to redirect the pages in a document to a particular paper tray or whether to scale pages or tile pages in a document.

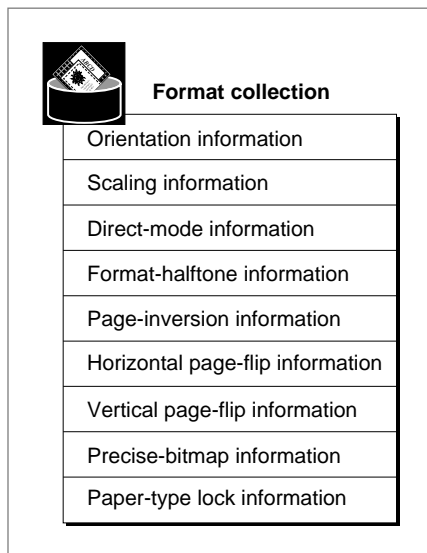
- **Tray-mapping information.** This collection specifies the tray-mapping information for the job object. It contains the index for a paper tray, which the user typically specifies by selecting a tray from the Paper Match panel of the Print dialog box.
- **Print-panel information.** This collection item contains the print-panel information for the job object. It specifies the name of the first panel to appear when your application displays the Print dialog box.
- **Format-panel information.** This collection item contains the format-panel information for the job object. It specifies the name of the first panel to appear when your application displays the Page Setup dialog box.
- **Paper-mapping information.** This collection item contains the paper-mapping information for the job object. If it is used, it contains a flattened paper-type resource.
- **Translated-document information.** This collection item contains the translated-document information for the job object. QuickDraw GX provides this information only for documents designed for printing with the Macintosh Printing Manager.

The Format Collection

QuickDraw GX primarily stores information from the Page Setup and Custom Page Setup dialog boxes in a format collection. You need to call the `GXChangedFormat` function each time you change the format collection.

QuickDraw GX stores nine items in a format collection, as shown in Figure 3-2.

Figure 3-2 The format collection



Page Formatting and Dialog Box Customization

A brief description of each collection item follows. To see how the pieces of data are structured in the collection item, see “Constants and Data Types for Format Collection Items” beginning on page 3-89. Format collection items include the following:

- **Orientation information.** This collection item contains the orientation information for the format object. It specifies whether to print a document (or a specific page) in portrait, landscape, or rotated landscape orientation. A user typically specifies orientation for an entire document in the Page Setup dialog box and specifies orientation for an individual page in the Custom Page Setup dialog box.
- **Scaling information.** This collection item contains the scaling information for the format object. It specifies a document’s horizontal and vertical scaling factors. It also stores the minimum and maximum scaling factors allowed. A user typically specifies scaling for an entire document in the Page Setup dialog box and specifies scaling for an individual page in the Custom Page Setup dialog box.
- **Direct-mode information.** This collection item contains the direct-mode information for the format object. It specifies whether the user chose the Direct checkbox in the Page Setup dialog box. (This checkbox appears only if the printer driver supports text job format mode printing.) The text job format mode is discussed in the chapter “Advanced Printing Features” in this book.
- **Format-halftone information.** This collection item contains the format-halftone information for the format object. It specifies the total number of halftone structures that can be used for a specific page and an array of halftone structures. You can use halftones to render continuous tone images on noncontinuous tone printers if the printer driver or a printing extension supports halftones. For an introduction to halftones, see the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*. For more information about this collection item, see “Halftones and Format Collections” beginning on page 3-21.
- **Page-inversion information.** This collection item contains the page-inversion information for the format object. It specifies whether to invert a page before printing.
- **Horizontal page-flip information.** This collection item contains the horizontal page-flip information for the format object. It specifies whether to horizontally flip the page left to right before printing.
- **Vertical page-flip information.** This collection item contains the vertical page-flip information for the format object. It specifies whether to vertically flip the page top to bottom before printing.
- **Precise-bitmap information.** This collection item contains the precise-bitmap information for the format object. It specifies whether to scale a page by 96% on 300-dpi printers.
- **Paper-type lock information.** This collection item contains the paper-type object lock information for the format object. It indicates whether the format’s paper-type object is locked.

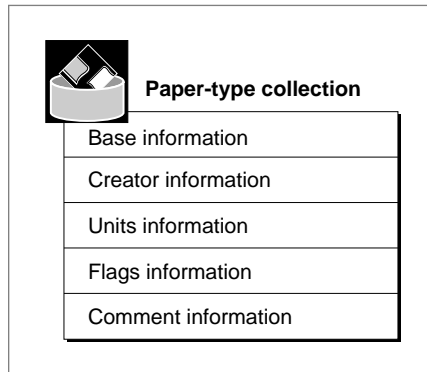
Note

The page-inversion information, page-flip information (horizontal and vertical), and precise-bitmap information are used, by default, only by PostScript printer drivers. ♦

The Paper-Type Collection

The paper-type collection contains additional information about the paper-type object. QuickDraw GX stores in a paper-type collection five collection items, as shown in Figure 3-3.

Figure 3-3 The paper-type collection



A brief description of each collection item follows. To see how the pieces of data are structured in the collection item, see “Constants and Data Types for Paper-Type Collection Items” beginning on page 3-94. Paper-type collection items include the following:

- **Base information.** This collection item contains the base paper type information for the paper-type object, which indicates the source from which the paper type was created. Base types include: unknown, US Letter, US Legal, A4, B5, and tabloid.
- **Creator information.** This collection item contains the creator information structure for the paper-type object. It specifies the creator type of a paper-type object; for example, 'sypt' for a system paper-type object creator and 'uspt' for a user paper-type object creator.
- **Units information.** This collection item contains the units information for the paper-type object. Units can be specified in picas, millimeters, and inches.
- **Flags information.** This collection item contains the flags information for the paper-type object. The flags are bits used to set or clear specific attributes of a paper-type object, such as whether the paper type is the default paper type for this format. For information about paper-type object flags, see “Flags Information” beginning on page 3-97.
- **Comment information.** This collection item contains the comment information for the paper-type object. It allows a comment to be associated with a paper-type object. You can specify application-specific information in this comment. For example, you may want to store a textual description of the paper-type and its purpose.

About Page Formatting

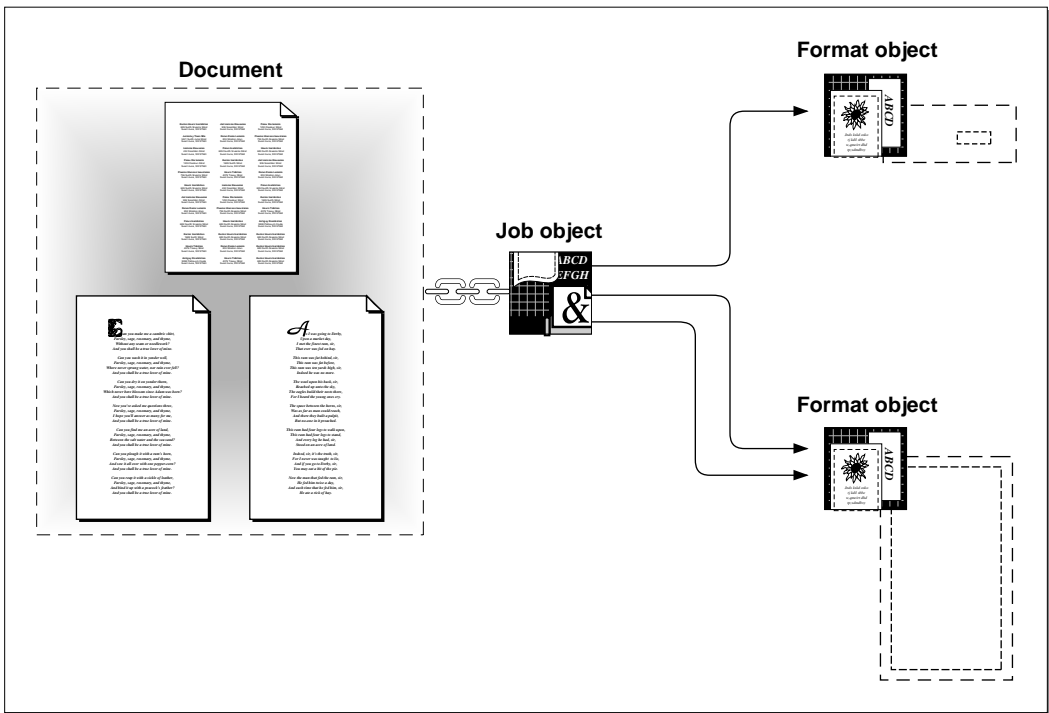
Page formatting allows the user to format specific pages of a document differently from the default formatting for the rest of the document. Using QuickDraw GX page formatting features, your application can

- allow users to specify unique formats for the individual pages of a document
- retrieve a format object's mapping
- attach a form to a format object as a backdrop to each page
- create documents that contain page-specific halftone information
- copy a format object for use in other documents

For example, using page-formatting features, a mail-merge application may automatically generate a document in which the first page consists of a template in which a user can enter addresses and the rest of the document consists of blank sheets in which a user can add text.

Figure 3-4 shows a document that is composed of a two-page letter and many address labels. The job object references two format objects, one for either page of the letter and the other for the address label.

Figure 3-4 A three page document and its corresponding job and format objects



Page Formatting and Dialog Box Customization

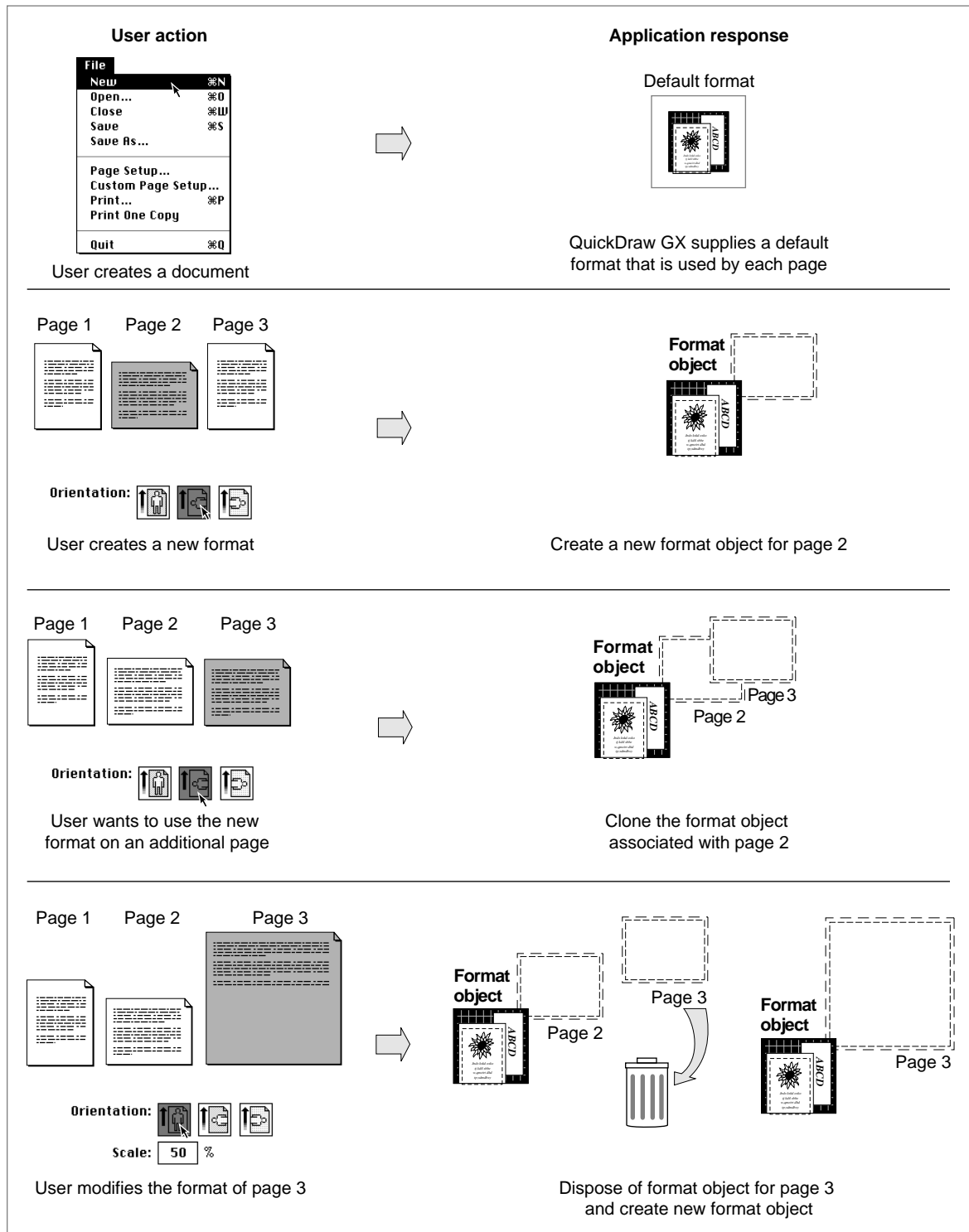
Manipulating format objects is described in the next section. Information on accessing a format object's mapping is discussed in "Mapping for Format Objects" beginning on page 3-18. Information on attaching a form to a format object is discussed in "Forms and Format Objects" beginning on page 3-20. Information on halftones is discussed in "Halftones and Format Collections" beginning on page 3-21.

Manipulating Format Objects

A format object contains the basic information that your application needs to display a single page or a set of pages. Generally, you work with format objects when a user

- creates a new format using the Custom Page Setup dialog box
- wants to use a format in several pages of a document
- modifies a format that is shared by other pages in the same document
- saves or opens a document

Figure 3-5 shows how you manipulate format objects in response to the first three actions.

Figure 3-5 Manipulating the format object in response to user actions

Page Formatting and Dialog Box Customization

When a user creates a new format through the Custom Page Setup dialog box, you need to create a new format object. Creating a new format object is discussed in “Creating a Format Object for a Page in a Document,” which begins on page 3-40.

Each format object you create has an associated owner count. The owner count indicates the number of times that a format object is shared. When a user creates a new format through the Custom Page Setup dialog box, you need to create a new format object with the `GXNewFormat` function. This function sets the owner count of a format object to 1.

When a user wants to use the new format to format another page the same way, you need to increment the format object’s owner count. You use the `GXCloneFormat` function to increment the owner count of a format object by 1. Cloning a format object is discussed in “Sharing Formats for Document Pages,” which begins on page 3-44.

When a user modifies a format object that is also shared by other pages, you need to dispose of its corresponding format object and create a new one. The `GXDisposeFormat` function decrements the owner count of a format object by 1. Disposing of a format object is discussed in “Disposing of a Format Object for a Page in a Document,” which begins on page 3-47.

To obtain the current owner count of a format object, you use the `GXCountFormatOwners` function. For more information about this function, see the description of `GXCountFormatOwners` on page 3-107.

You also must create a correspondence between the format and the page. Typically, you keep the correspondence in the format collection. You must save the correspondences when the job is flattened and retrieve them when the job is unflattened. For an example, see “Associating Format Objects With Document Pages” beginning on page 3-61.

Mapping for Format Objects

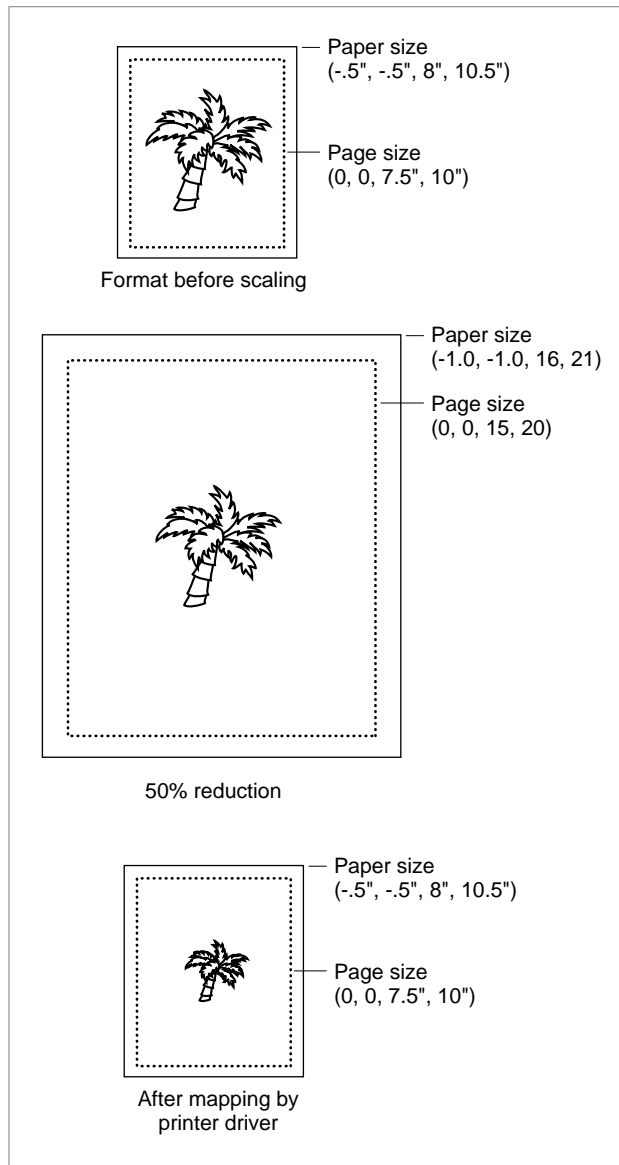
A format object’s mapping is a mathematical representation of the format object’s settings. These settings include the paper size, page size, orientation, and scaling. The paper size and page size are set when you create the format object.

QuickDraw GX uses this mapping to scale page information into device pixels. A device pixel is the smallest physical area that a printer is capable of rendering. Typically, the mapping consists of the high-resolution scaling information needed to print a page at the highest quality.

QuickDraw GX and the printer driver set up the mapping. Your application can retrieve the mapping but cannot set it directly. You might want to retrieve it, for example, to set the mapping property of a view port to represent the printer on screen. For more information about view port objects, see the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

Figure 3-6 shows how the scaling item affects the mapping.

Figure 3-6 Scaling a format object



When 50% scaling is applied, the scaling variables in the mapping are actually doubled, which causes the shape to appear the same size on a page of paper that is twice its original size. When the printer driver maps the page to dots-per-inch, it reduces the format dimension and everything within them, including the shape object. The result is that the shape is scaled to 50% when it is printed.

Forms and Format Objects

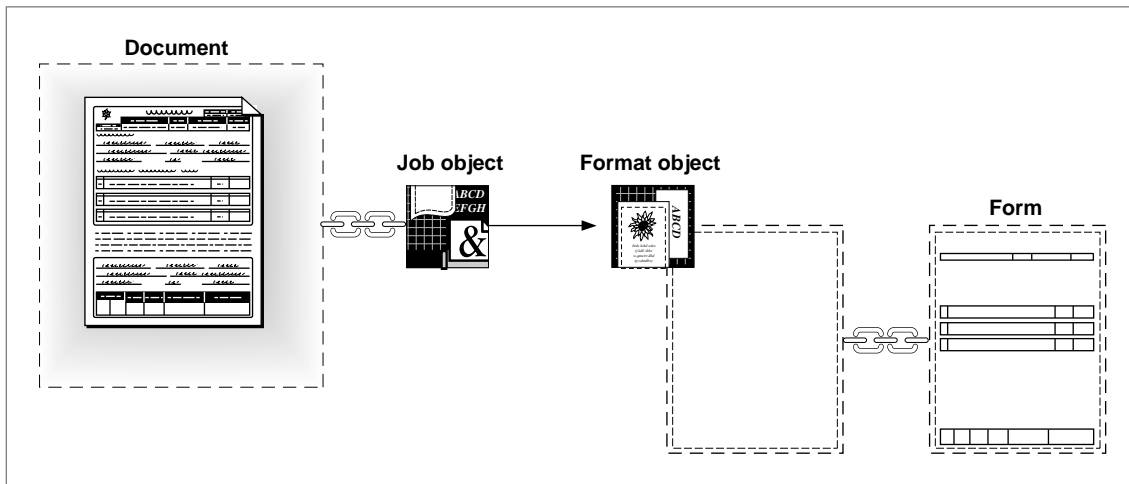
QuickDraw GX provides the form property, which allows your application to format pages of a document with a template. A form is made up of two shape objects—one shape defines the form itself, and the other shape, the mask shape, defines erasable areas within the form. The mask shape is optional; your application can erase the contents within a form, but this technique is not recommended.

Your application can specify a form for any format object associated with the formatting printer. Your application uses this form as a backdrop that is applied to a set of pages. For example, you can use a form to define erasable areas within pages created using a database application.

To associate a form shape and a mask shape with a format object associated with a page, you use the `GXSetFormatForm` function. To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function. The shape type that you associate with a format object must be a picture shape.

Figure 3-7 shows a page from a document created by a database application. The figure also shows the job object corresponding to the document, the job's format object, and a form.

Figure 3-7 Using a form to format a page



Forms save time during spooling, rendering, and I/O. During spooling, QuickDraw GX spools a form shape only once. QuickDraw GX renders a form shape once for each distinct format object it is attached to. During I/O, if the printer can cache the representation of the form, QuickDraw GX saves data transmission time by sending the form to the printer only when it has to.

Halftones and Format Collections

You can use a halftone to represent more colors than can be represented on a printer by alternating available colors of a fixed cell size so that a noncontinuous tone device appears to produce continuous-tone grayscale or full-color images.

You can use the format collection to specify halftone information on a page-by-page basis. Initially, the printer driver specifies the halftone information for the default format by storing the information in this format's format collection object. You can add halftones to this collection, in which case you are changing the halftone for the entire document. You can also change the halftone information in a format collection associated with the format object for specific pages, in which case only the pages associated with the format object receive the halftone. Storing halftone information in a format collection is discussed in "Storing Halftone Information in a Format Collection," which begins on page 3-52.

Note

To specify halftone information on a shape-by-shape basis, you use a synonym attached to the shape's ink object. For more information about the halftone synonym, see the chapter "Advanced Printing Features" in this book. ♦

The format-halftone item in the format collection specifies the halftones to use. The collection item specifies a `gxFormatHalftoneInfo` structure that defines the number of allowable halftones and their characteristics. For the definition of the `gxFormatHalftoneInfo` structure, see "Format-Halftone Information" on page 3-92.

The definition of each halftone is specified in a `gxHalftone` structure, which is described completely in the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*:

```
struct gxHalftone{
    fixed          angle;           /* direction of halftone */
    fixed          frequency;       /* cells per inch */
    gxDotType      method;          /* kind of pattern */
    gxTintType     tinting;         /* tint calculation method */
    gxColor        dotColor;        /* color of foreground */
    gxColor        backgroundColor; /* color of background */
    gxColorSpace   tintSpace;       /* color space for tint */
};
```

You can specify any number of these `gxHalftone` structures in the format-halftone information item. QuickDraw GX selects appropriate halftones from the list of available halftones in the item. Its selection is based upon the `tinting` field in the halftone structure:

- When you print to a black-and-white PostScript device, QuickDraw GX looks for a halftone structure that specifies `gxLuminanceTint` in the `tinting` field. If no halftone specifies this value, it looks for a halftone that specifies `gxComponent4Tint` as its tinting method. Component 4 is the black component in the CMYK (cyan,

magenta, yellow, and black) space. If no halftone specifies this tinting method either, the first halftone in the list is used.

- When you print to a color PostScript device, a maximum of four halftones are used. QuickDraw GX attempts to locate halftones for the following tint calculation methods: `gxComponent1Tint` for the cyan halftone, `gxComponent2Tint` for the magenta halftone, `gxComponent3Tint` field for the yellow halftone, and `gxComponent4Tint` for the black halftone. If a tinting method is in the list more than once, the first one in the list is used.

If a halftone for the `gxComponent4Tint` method is not in the list, QuickDraw GX uses the `gxLuminanceTint` tinting method for the black halftone. If the `gxLuminanceTint` tinting method cannot be found either, QuickDraw GX uses the first halftone in the list for the black halftone.

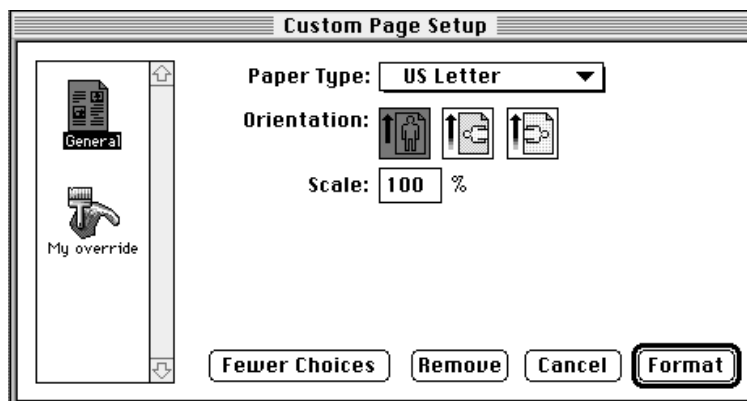
If QuickDraw GX cannot find a halftone for the `gxComponent1Tint`, `gxComponent2Tint`, or `gxComponent3Tint` tinting methods, it uses the black halftone for the missing tinting method.

It is only possible to use halftones to the extent that a particular PostScript device supports them. The dot color and background color of a halftone are ignored because QuickDraw GX assumes that the dot color for a black-and-white device is black and the dot color for a color device with the `gxComponent2Tint` tinting method is magenta.

Dialog Box Customization

QuickDraw GX allows your application to customize print dialog boxes, typically, by adding panels. A panel is a portion of an expanded dialog box that presents additional printing options for users. For example, you may allow the user to specify custom margins in a panel you add to the Page Setup dialog box or the Custom Page Setup dialog box. Figure 3-8 shows the expanded Custom Page Setup dialog box with two panels, the General panel and the “My override” panel. The contents of the General panel are shown in Figure 3-8.

Figure 3-8 The expanded Custom Page Setup dialog box with two panels



QuickDraw GX stores a user's responses to most default dialog items in collection objects. Your application can use collection objects to store information from panels you have added to dialog boxes. For information about storing items in collection objects and retrieving them, see "Using Printing-Related Collection Objects" beginning on page 3-27.

Note

If several applications want to provide the same option in the same panel, it may be better to implement the panel in a printing extension. For more information about printing extensions, see the printing extensions chapter of *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*. ♦

To create a panel, you must define a panel resource (`gxPrintPanelType`), as described in the next section. You may also define an extended item list resource (`gxExtendedDITLType`) that defines how to respond to user actions, such as clicking a button, while the panel is on the screen. This resource is described in the section "Automating Panel Events" beginning on page 3-25.

Messages are used to notify the application when a print dialog box is about to be displayed. This allows you to load the panel from the resource before the dialog box is displayed. The functions that invoke these messages are shown in Table 3-1.

Table 3-1 Functions that enable dialog box panels

Function	Description
<code>GXJobPrintDialog</code>	Displays the Print dialog box
<code>GXJobDefaultFormatDialog</code>	Displays the Page Setup dialog box
<code>GXFormatDialog</code>	Displays the Custom Page Setup dialog box

Your application typically takes these steps to enable a panel when the user chooses a menu item that brings up a dialog box:

1. Call the appropriate function, such as `GXJobPrintDialog` if the user chose the Print menu item from the File menu.
2. Respond to the message, such as `gxJobPrintDialog`, by invoking your override function; for example, `MyJobPrintDialog`. This response was set up by the call to `GXInstallApplicationOverride` to set up the application as a message handler.
3. Set up the panel and call `GXSetupDialogPanel` to display it. These actions are performed by the override function.
4. Forward the message. This action is also performed by the override function.

For an overview of how messages allow you to display a panel in a print dialog box, see the chapter "Introduction to Printing With QuickDraw GX" in this book.

To forward a message, you call one of the functions in Table 3-2.

Table 3-2 Functions that forward a dialog box message

Function	Description
Forward_GXJobPrintDialog	Forwards the <code>gxJobPrintDialog</code> message
Forward_GXJobDefaultFormatDialog	Forwards the <code>gxJobDefaultFormatDialog</code> message
Forward_GXFormatDialog	Forwards the <code>gxFormatDialog</code> message

You pass exactly the same arguments to the forwarding function as those with which your override function was called. For an example of setting up a custom dialog box with an added panel and forwarding a message, see the section “Adding Panels to Dialog Boxes” beginning on page 3-67. For information about how to forward other messages, see the Message Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

The Dialog Box Panel Resource

A panel resource defines a panel. It specifies the following information:

- The panel’s name, such as `My override`. The name appears in the list of panels for an extended dialog box, under the panel’s icon.
- The script used to display the panel, such as `smRoman`.
- The resource of the ‘DITL’ resource that defines the items in the panel.
- The resource ID of the icon to display in the extended dialog box.

Listing 3-1 shows the structure of a panel resource.

Listing 3-1 A panel resource definition template

```
type gxPrintPanelType {
    pstring[31];    /* name */
    integer Script; /* international script */
    fill word;      /* long word reserved for future use */
    fill word;      /* long word reserved for future use */
    integer;        /* the icon id */
    integer;        /* the ditl id */
};
```

Responding to Panel Events

QuickDraw GX handles events for its default panels automatically. If you change a default panel or you add another one, you may need to override the messages that QuickDraw GX sends in order to process the items that you added.

If an event occurs while a panel is displayed, QuickDraw GX sends a `gxFilterPanelEvent` message. If you want to filter the event, you can override this message by installing a handler for it and by specifying a function that matches the prototype defined for `GXFilterPanelEvent` on page 3-124.

If you do not need to filter the event, you may choose to handle the event in your code, you may automate the handling of the event, or you may do both. Events that you need to handle in some way include mouse clicks on radio buttons or checkboxes, choosing an item from a pop-up menu, and keystrokes in editable text fields.

To handle the event in your application code, you install an override for the `gxHandlePanelEvent` message. You can override this message by installing a handler for it and specifying a function that matches the prototype defined for `GXHandlePanelEvent` on page 3-123.

For information about messages and how to override them, see the chapter “Introduction to Printing With QuickDraw GX” in this book. For an example of installing an override function, see the chapter “Core Printing Features” in this book. For information about automatically handling panel events, see the next section.

Automating Panel Events

You can allow QuickDraw GX to automatically respond to selections in dialog box panels without you writing additional application code. QuickDraw GX provides an extended item list (`gxExtendedDITLType`) resource that loads values or settings of items and responds to changes to items in an item list (‘DITL’) resource. The item types for which QuickDraw GX can load values or settings and respond to changes in them include

- radio buttons
- checkboxes
- pop-up menus
- editable text in strings; the strings may represent characters, integers, and real numbers

QuickDraw GX obtains the values or settings from items in the job and format collections and responds to changes, by updating the information in these items, when the changes are confirmed. If the panel is part of the Print dialog box, the collection item must be in the job collection. If the panel is part of the Page Setup or Custom Page Setup dialog box, the collection item must be in the format collection.

For example, as a panel is displayed, an extended item resource specifies the collection item to use to set a group of radio buttons. If a user clicks on an unselected radio button, QuickDraw GX deselects the previously highlighted button and highlights the chosen

one. When the user closes the panel and confirms the settings (for example, by clicking the OK button), the items specified by the extended item resource are placed back in their collections. If the user cancels the panel, the collection items are not changed.

The processing for checkboxes is similar. If the checkbox is not checked, QuickDraw GX checks it; if it is checked, QuickDraw GX unchecks it. Editable text is checked when the panel is closed and confirmed.

QuickDraw GX uses the resource IDs of the extended item list resource and 'DITL' resources to determine which extended item list to associate with the item list. If both kinds of resources have the same ID, they are used together. Specifically, when an open-panel (`gxPanelOpenEvt`) message is sent in response to the user choosing a panel in a dialog box, QuickDraw GX uses the extended item list resource that corresponds to the panel's 'DITL' resource to load and set the items. (Recall from the previous section that the panel resource specifies a 'DITL' resource.)

Listing 3-2 shows the extended item resource definition for editable text that represents a real number.

Listing 3-2 The extended item list resource definition template

```
#define  xdtlRadioButtons      0
#define  xdtlCheckBox         1
#define  xdtlEditTextInteger  2
#define  xdtlEditTextReal     3
#define  xdtlEditTextString   4
#define  xdtlPopUp            5

type gxExtendedDITLType {
...
    case EditTextReal:
        key      integer = xdtlEditTextReal;
        literal  longint; /* 4 byte id for storage in job
                           object or format object */
                           longint; /* numerical id for storage in
                           job object or format object */
        integer; /* offset in bytes into the item
        byte;    /* corresponding ditl item */
        byte;    /* 0 = dont select, 1 = select
        pstring[15];/* low bound - nil means 'I
                           don't care' */
        pstring[15];/* high bound - nil means 'I
                           don't care' */
...
    };
};
```

There are common fields for each item type supported by an extended item list resource:

- The tag ID, such as `gxPrintingTagID` for QuickDraw GX defined tags.
- The ID of the collection item, such as `gxFormatHalftoneTag`.
- The offset into the collection item. The offset allows you to specify several values, such as the settings for checkboxes, in the same item.
- The corresponding item in the 'DITL' resource, starting with 1.

The remaining fields depend on the kind of data. For real number editable text, the fields specify the following:

- Whether or not to highlight the field's contents when the panel is displayed; 0 specifies do not highlight, 1 specifies to highlight.
- The lowest possible value for range checking. A `nil` string specifies no lower bound.
- The highest possible value for range checking. A `nil` string specifies no upper bound.

If a user enters data that does not conform to the specified format or specifies a number that is out of range, the text item inverts, and a system beep alerts the user to the problem when the user attempts to leave the field.

For the definitions of other kinds of fields, see "The Extended Item List Resource" on page 3-128. For an example of specifying an extended item list resource, see "Setting Up Dialog Box Resources" on page 3-70.

Using Printing-Related Collection Objects

Your application can use collection objects to store information associated with a particular document. To access collection objects used by QuickDraw GX printing features, you use functions provided by QuickDraw GX. You manipulate the pieces of information in collection objects using Collection Manager functions. The Collection Manager is described in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

QuickDraw GX allows you to access a collection object associated with a job object, format object, or paper-type object. If you want to store or access printing-related information associated with a document in the job collection, you use the `GXGetJobCollection` function to access this collection object. If you want to store or access formatting information in the format collection, you use the `GXGetFormatCollection` function. If you want to store or access paper-type information in the paper-type collection, you use the `GXGetPaperTypeCollection` function.

Page Formatting and Dialog Box Customization

You then specify the collection item in a call to `GetCollectionItem` to retrieve the specific data from a collection. The collection item corresponds to the data you wish to retrieve. For example, the `gxCopiesTag` collection item specifies access to data in the `gxCopiesInfo` data structure:

```
enum {gxCopiesTag = 'copy'};

struct gxCopiesInfo{
    long copies;
};
```

Note

The collection tags, collection item tags, and structures for collection objects are defined in the section “Constants and Data Types” beginning on page 3-133. For complete information about using collections, see the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. ♦

Accessing Data From a Collection Object

The following example shows you how to access an item from a collection object. For an example of adding an item to a collection object, see “Associating Format Objects With Document Pages” on page 3-61. For an example of replacing a collection item, see “Using a Collection to Implement the Print One Copy Menu Item” on page 3-29.

Listing 3-3 shows how to use the `GXGetJobCollection` function to access the number of copies, which is stored in a job collection.

Listing 3-3 Accessing copies information stored in a job collection

```
OSErr MyGetJobCopies(MyDocumentPtr myDocument, long *numCopies)
{
    OSErr          err;
    Collection      jobCollection;
    gxCopiesInfo    theCopiesInfo;
    long            dataSize = sizeof(theCopiesInfo);

    /*
       Get the job collection and look for a gxCopiesTag collection
       object item.
    */
    jobCollection = GXGetJobCollection(myDocument->documentJob);
    err = GetCollectionItem(jobCollection,
                           gxCopiesTag,
```



```

                                gxPrintingTagID,
                                dataSize,
                                &theCopiesInfo);

    /* Extract the number of copies and return it. */
    if (err == noErr)
        *numCopies = theCopiesInfo.copies;
    return err;
}

```

Using a Collection to Implement the Print One Copy Menu Item

To implement the Print One Copy menu item, you must change items in the job collection. You must ensure that only one copy will be printed, that all pages will be printed, and that the output will actually go to a printer and not to a print file. After the copy has been printed, you must set the contents of the collection items back to their original values so that the user's settings are preserved.

Listing 3-4 shows how to set the values of the `gxCopiesTag`, `gxPageRangeInfo`, and `gxFileDestinationTag` items so that only one copy of all pages is printed and it is sent to the printer. It also shows how to restore the original values for these collection items after the print operation has been completed.

Listing 3-4 Modifying the job collection to implement the Print One Copy menu item

```

OSErr MyPrintOneCopy(MyDocumentPtr whichDocument)
{
    OSERR                err;
    Collection            jobCollection;
    gxCopiesInfo          copiesInfo;
    gxFileDestinationInfo destInfo;
    gxPageRangeInfo       pageRangeInfo;
    Ptr                   oldCopiesInfo = nil;
    Ptr                   oldPageRangeInfo = nil;
    Ptr                   oldDestInfo = nil;
    long                  oldCopiesSize;
    long                  oldPageRangeInfoSize;
    long                  oldDestInfoSize;

    /* Get the job collection and set it up to print one copy */
    jobCollection = GXGetJobCollection(whichDocument->documentJob);

```

Page Formatting and Dialog Box Customization

```

/* Set number of copies to 1 */
copiesInfo.copies = 1;
err = MyReplaceCollectionItem(&copiesInfo,
                             sizeof(gxCopiesInfo),
                             gxCopiesTag, gxPrintingTagID,
                             jobCollection, &oldCopiesInfo,
                             &oldCopiesSize);
nrequire(err, ReplaceCopies_error);

/* Set page range to "all". */
pageRangeInfo.simpleRange.optionChosen = gxDefaultPageRange;
pageRangeInfo.minFromPage = 1;
pageRangeInfo.simpleRange.fromPage = 1;
pageRangeInfo.maxToPage = whichDocument->numPages;
pageRangeInfo.simpleRange.toPage = whichDocument->numPages;
pageRangeInfo.simpleRange.printAll = true;
err = MyReplaceCollectionItem(&pageRangeInfo,
                             sizeof(gxPageRangeInfo),
                             gxPageRangeTag, gxPrintingTagID,
                             jobCollection, &oldPageRangeInfo,
                             &oldPageRangeInfoSize);
nrequire(err, ReplacePageRange_error);

/* Set destination to "printer". */
destInfo.toFile = false;
err = MyReplaceCollectionItem(&destInfo,
                             sizeof(gxFileDestinationInfo),
                             gxFileDestinationTag, gxPrintingTagID,
                             jobCollection, &oldDestInfo,
                             &oldDestInfoSize);
nrequire(err, ReplaceDestination_error);

/* Print one copy of the document. (not shown here) */
err = MyPrintDocument(whichDocument);

/*
   Restore original number of copies, page range, and output
   destination in case it needs to be reused.
*/
ReplaceCopies_error:
    MyReplaceCollectionItem(oldCopiesInfo, oldCopiesSize,
                           gxCopiesTag, gxPrintingTagID,
                           jobCollection, nil, nil);

```

```

ReplacePageRange_error:
    MyReplaceCollectionItem(oldPageRangeInfo, oldPageRangeInfoSize,
                           gxPageRangeTag, gxPrintingTagID,
                           jobCollection, nil, nil);
ReplaceDestination_error:
    MyReplaceCollectionItem(oldDestInfo, oldDestInfoSize,
                           gxFileDestinationTag, gxPrintingTagID,
                           jobCollection, nil, nil);

    /* Dispose of pointers created by MyReplaceCollectionItem */
    if (oldCopiesInfo)
        DisposePtr(oldCopiesInfo);
    if (oldPageRangeInfo)
        DisposePtr(oldPageRangeInfo);
    if (oldDestInfo)
        DisposePtr(oldDestInfo);

    return err;
}

```

Replacing Items in Collections

The `MyReplaceCollectionItem` function is a generic routine that you could write to replace collection items. In the implementation in Listing 3-5, the data being replaced is returned in a variable pointed to by `oldData`, unless the pointer is `nil`. If the item does not exist, the new data is returned via the pointer instead.

Listing 3-5 Replacing collection items

```

OSError MyReplaceCollectionItem(void *newData, long collectSize,
                                OSType collectType, long collectID,
                                Collection whichCollection,
                                Ptr *oldData, long *oldDataSize)
{
    OSError    err;
    long       index;

    /*
     * If returning the old data, get it.
     * If there is no old data, return a copy of the new data.
     */
    if (oldData)

```

Page Formatting and Dialog Box Customization

```

{
    err = GetCollectionItemInfo(whichCollection,
                                collectType,
                                collectID,
                                dontWantIndex,
                                oldDataSize,
                                dontWantAttributes);

    if (err)
    {
        *oldDataSize = collectSize;
        *oldData = NewPtrSys(*oldDataSize);
        if (!(err = MemError()))
            BlockMove(newData, *oldData, collectSize);
    }
    else
    {
        *oldData = NewPtrSys(*oldDataSize);
        if (!(err = MemError()))
            err = GetCollectionItem(whichCollection,
                                    collectType,
                                    collectID,
                                    dontWantSize,
                                    *oldData);
    }
    nrequire(err, CouldNotSetOldData);
}

/*
    Add a new collection item; otherwise, get the existing
    item's index and replace the old collection item.
*/
err = AddCollectionItem(whichCollection,
                        collectType,
                        collectID,
                        collectSize,
                        newData);

```

```

if (err == collectionItemLockedErr)
{
    err = GetCollectionItemInfo(whichCollection,
                                collectType,
                                collectID,
                                &index,
                                dontWantSize,
                                dontWantAttributes);

    if (!err)
        err = ReplaceIndexedCollectionItem(whichCollection,
                                            index,
                                            collectSize,
                                            newData);
}

CouldNotSetOldData:
    return err;
}

```

Specifying Page Ranges in the Job Collection

You can specify the page range and page range constraints in the page-range information job collection item. QuickDraw GX provides three kinds of representations for page ranges: simple numeric From and To values called the default page range, a single editable text field that specifies a replacement page range, and alphanumeric From and To values called a customized page range.

Listing 3-4 on page 3-29 shows how to set up a default page range for all pages to support the Print One Copy menu item of the File menu. The examples that follow show how to set up default, replacement, and customized page information for specific pages.

Listing 3-6 on page 3-33 shows how to set up the default page range for pages 1 through 4. The user may change these values to any within 1 and 9999.

Listing 3-6 Setting up a default page range

```

OSErr MyConfigurePageRange1(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxPageRangeInfo **pageRangeHdl;

    /*
     * Create a handle to store the page range collection item in,
     * and then retrieve the collection item.
     */
}

```

Page Formatting and Dialog Box Customization

```

pageRangeHdl
= (gxPageRangeInfo **)NewHandleClear(sizeof(gxPageRangeInfo));
nrequire_action(err, NewHandleClear_Failed, err = MemError());

err = GetCollectionItemHdl
      (GXGetJobCollection(myDocument->documentJob),
       gxPageRangeTag,
       gxPrintingTagID,
       (Handle) pageRangeHdl);
nrequire(err, GetCollectionItemHdl_Failed);

/*
   Use the standard "From-To" editable text field containing
   default numeric values.
   Specify that the "all pages" radio button is not to be
   selected and that the "From" field contains 1 and the
   "To" field contains 4.
*/
(*pageRangeHdl)->simpleRange.optionChosen = gxDefaultPageRange;
(*pageRangeHdl)->simpleRange.printAll = false;

(*pageRangeHdl)->simpleRange.fromPage = 1;
(*pageRangeHdl)->simpleRange.toPage = 4;
(*pageRangeHdl)->minFromPage = 1;
(*pageRangeHdl)->maxToPage = 9999;

/* Add (or replace) the collection item, and dispose of its
   handle. */
err = AddCollectionItemHdl(
      GXGetJobCollection(myDocument->documentJob),
      gxPageRangeTag,
      gxPrintingTagID,
      (Handle) pageRangeHdl);

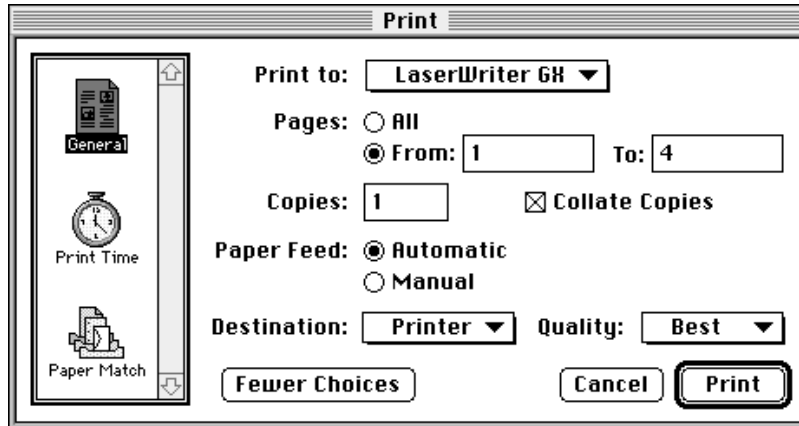
GetCollectionItemHdl_Failed:
    DisposHandle((Handle) pageRangeHdl);

NewHandleClear_Failed:
    return err;
}

```

Figure 3-9 shows the Print dialog box after executing the code to set the page range. QuickDraw GX obtains the page range to display from the collection item.

Figure 3-9 Print dialog box with default page range



Listing 3-7 shows how to set up a replacement page range, in which the From and To fields are replaced by a single editable text field. Note that the default editable text field is only one character, therefore, you almost always increase the size of the handle to accommodate the page range. The page range is a Pascal-style string.

Listing 3-7 Setting up a replacement page range

```
OSErr MyConfigurePageRange2(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxPageRangeInfo **pageRangeHdl;

    /*
       Create a handle to store the page range collection item in,
       and then retrieve the collection item.
    */
    pageRangeHdl
        = (gxPageRangeInfo **)NewHandleClear(sizeof(gxPageRangeInfo));
    nrequire_action(err, NewHandleClear_Failed, err = MemError());
}
```

Page Formatting and Dialog Box Customization

```

err = GetCollectionItemHdl
    (GXGetJobCollection(myDocument->documentJob),
     gxPageRangeTag,
     gxPrintingTagID,
     (Handle) pageRangeHdl);
nrequire(err, GetCollectionItemHdl_Failed);

/*
    Replace the standard "From-To" editable text fields, with a
    single editable text field that contains "Chapter 5."
    Specify that the "all pages" radio button is not to be
    selected.
*/
(*pageRangeHdl)->simpleRange.optionChosen = gxReplacePageRange;
(*pageRangeHdl)->simpleRange.printAll = false;
SetHandleSize((Handle) pageRangeHdl,
    sizeof(gxPageRangeInfo)+titleSize-1);
nrequire_action(err, SetHandleSize_Failed, err = MemError());
BlockMove(FromToTitle, (*pageRangeHdl)->replaceString,
    titleSize);

/* Add (or replace) the collection item, and dispose of its
   handle. */
err = AddCollectionItemHdl(
    GXGetJobCollection(myDocument->documentJob),
    gxPageRangeTag,
    gxPrintingTagID,
    (Handle) pageRangeHdl);

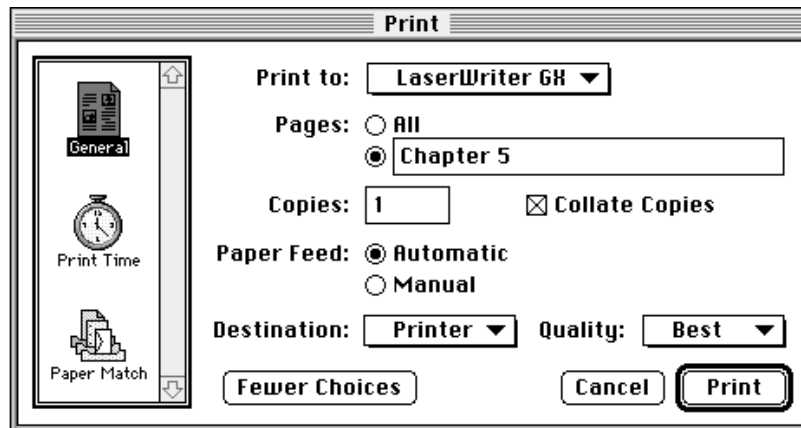
SetHandleSize_Failed:
GetCollectionItemHdl_Failed:
    DisposHandle((Handle) pageRangeHdl);

NewHandleClear_Failed:
    return err;
}

```


Figure 3-10 shows the Print dialog box after executing the replacement page range code. The contents of the title, “Chapter 5,” are displayed in a single editable text field. You must check for the validity of this field if the user changes it. For more information about parsing a page range to test its validity, see “Parsing Page Ranges” on page 3-73.

Figure 3-10 Print dialog box with replacement page range



Listing 3-8 shows how to set up a customized page range, in which the From and To fields allow editable text.

Listing 3-8 Setting up a customized page range

```
OSErr MyConfigurePageRange3(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxPageRangeInfo **pageRangeHdl;

    /*
       Create a handle to store the page range collection item in,
       and then retrieve the collection item.
    */
    pageRangeHdl
        = (gxPageRangeInfo **)NewHandleClear(sizeof(gxPageRangeInfo));
    nrequire_action(err, NewHandleClear_Failed, err = MemError());
```

Page Formatting and Dialog Box Customization

```

err = GetCollectionItemHdl
        (GXGetJobCollection(myDocument->documentJob),
         gxPageRangeTag,
         gxPrintingTagID,
         (Handle) pageRangeHdl);
nrequire(err, GetCollectionItemHdl_Failed);

/*
    Use the standard "From-To" editable text fields, but they
    now contain a custom format for the page range values.
    Specify that the "all pages" radio button is not to be
    selected and that the "From" field contains "iii" and the
    "To" field contains "VI".
*/
(*pageRangeHdl)->simpleRange.optionChosen =
                                                gxCustomizePageRange;
(*pageRangeHdl)->simpleRange.printAll = false;
BlockMove("iii", &(*pageRangeHdl)->fromString[1], 3);
(*pageRangeHdl)->fromString[0] = 3;
BlockMove("VI", &(*pageRangeHdl)->toString[1], 2);
(*pageRangeHdl)->toString[0] = 2;

/* Add (or replace) the collection item, and dispose of its
   handle. */
err = AddCollectionItemHdl(
        GXGetJobCollection(myDocument->documentJob),
        gxPageRangeTag,
        gxPrintingTagID,
        (Handle) pageRangeHdl);

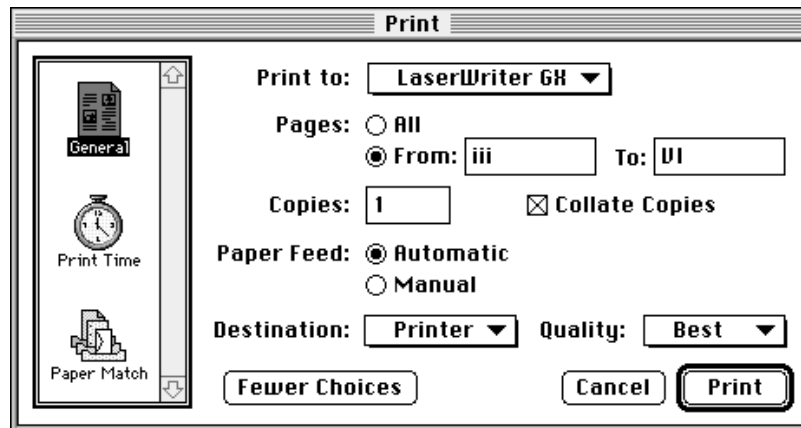
GetCollectionItemHdl_Failed:
    DisposHandle((Handle) pageRangeHdl);

NewHandleClear_Failed:
    return err;
}

```

Figure 3-11 shows the Print dialog box after executing the customized page range code. The contents of the From and To fields are now editable text. You must check for the validity of these fields if the user changes them. For more information about parsing a page range to test its validity, see “Parsing Page Ranges” on page 3-73.

Figure 3-11 Print dialog box with customized page range



Using Format Objects and Collection Items to Format Pages

To support page-formatting features, your application needs to manipulate format objects and keep track of the number of times a format object is shared. Generally, you work with format objects when a user creates a new format, wants to share a format with additional pages in a single document, disposes of a page, or modifies a format that is shared by other pages in the same document. Because your application is responsible for associating format objects with the individual pages of a document, you need to save this association when a user saves a document.

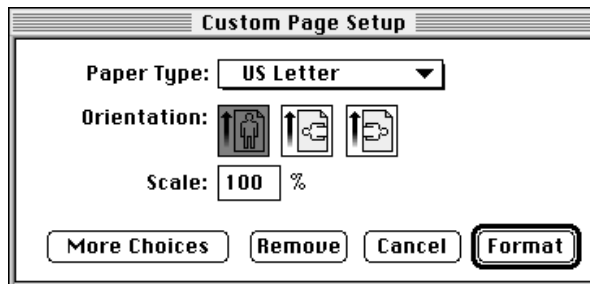
Your application can also manipulate format objects to support special formatting features. These features include associating form shapes with format objects, supporting page-specific halftone information in your application's documents, and copying format objects for use in multiple documents. QuickDraw GX also allows you to access information associated with a specific format object, such as its mapping and associated paper-type objects.

The following sections describe how to use page-formatting features.

Creating a Format Object for a Page in a Document

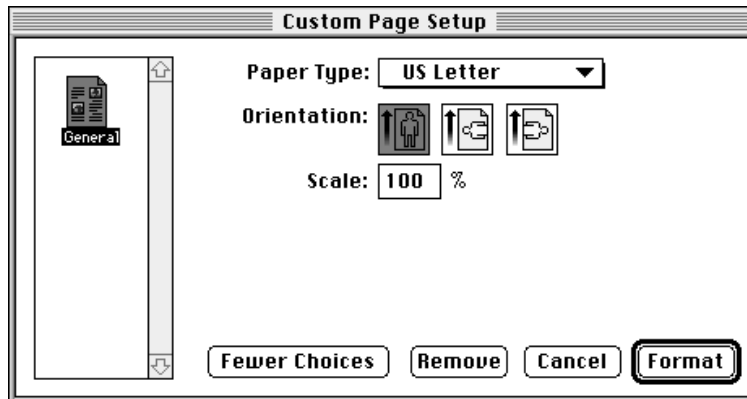
When a user wants to create a unique format or change its settings, the user chooses the Custom Page Setup menu item from the File menu. In response, you need to call the `GXFormatDialog` function to display the Custom Page Setup dialog box on the user's screen. Figure 3-12 shows the Custom Page Setup dialog box.

Figure 3-12 The Custom Page Setup dialog box



If the user clicks the More Choices button in the Custom Page Setup dialog box, QuickDraw GX expands the dialog box. Figure 3-13 shows the expanded Custom Page Setup dialog box.

Figure 3-13 The expanded Custom Page Setup dialog box

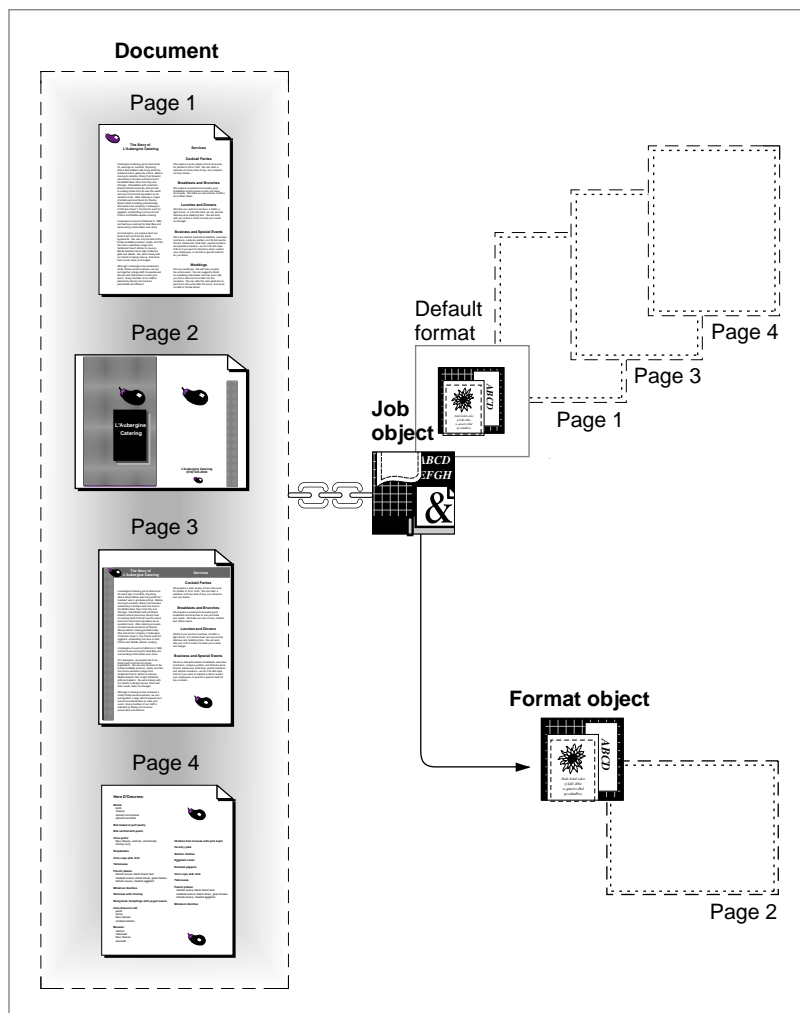


You need to create a new format object when a user chooses the Format button in the Custom Page Setup dialog box. For example, a user may create a four-page document, move to page 2, and then choose landscape orientation in the Custom Page Setup dialog box. The change to the page occurs when the user chooses the Format button.

Page Formatting and Dialog Box Customization

Figure 3-14 shows a four-page document in which the second page uses a new format. Pages 1, 3, and 4 use the default format. Pages 1, 3, and 4 use the default format.

Figure 3-14 A four-page document in which page two uses a unique format object



Page Formatting and Dialog Box Customization

Listing 3-9 shows how to create a new format object for a single page in a document. You should note that the `GXNewFormat` function sets the owner count for the new format object to 1.

Listing 3-9 Creating a format object for a page in a document

```
OSErr MyPageFormatDialog(MyDocumentPtr myDocument)
{
    OSErr          err = noErr;
    gxDialogResult  result;
    gxEditMenuRecord editMenuRec;
    gxFormat        pageFormat;
    Boolean         newPgFormat = false;

    /* Fill in the location of your application's Edit menu items. */
    editMenuRec.editMenuID = mEdit;
    editMenuRec.cutItem     = kCut;
    editMenuRec.copyItem    = kCopy;
    editMenuRec.pasteItem   = kPaste;
    editMenuRec.clearItem   = kClear;
    editMenuRec.undoItem    = kUndo;

    /* Modify existing format object, else create a new one. */
    if (myDocument->pageFormat[myDocument->curPage -1] != nil)
        pageFormat = myDocument->pageFormat[myDocument->curPage -1];
    else
    {
        pageFormat = GXNewFormat(myDocument->documentJob);
        newPgFormat = true;
        err = GXGetJobError(myDocument->documentJob);
    }

    /* If no errors, display the Page Setup dialog box. */
    if (err == noErr)
    {
        result = GXFormatDialog(pageFormat, &editMenuRec, nil);
    }
}
```

Page Formatting and Dialog Box Customization

```

/*
    If the user chooses Remove, use the default format for
    this page. If the user chooses Format, store the new
    format with this page. If the user chooses Cancel,
    dispose of the cloned copy of the default format.
*/
switch (result)
{
    case gxRevertSelected:
        GXDisposeFormat(pageFormat);
        pageFormat = nil;

    case gxOKSelected:
        myDocument->pageFormat[myDocument->curPage -1] =
            pageFormat;

        /*
            Place code here if your application needs to
            adjust the document based on the new format object.
        */
        ...
        break;

    case gxCancelSelected:
        /*
            If the user selects Cancel, dispose of the cloned
            copy of the default format object.
        */
        if (newPgFormat) GXDisposeFormat(pageFormat);
        break;
    }
}
return err;
}

```

Page Formatting and Dialog Box Customization

Within the Custom Page Setup dialog box used to create unique formats, the user has the option to return to the default format by choosing the Remove button or to not save the format by choosing the Cancel button. These options are handled by the `gxRevertSelected` and `gxCancelSelected` cases, respectively, of the switch statement in Listing 3-9.

For example, if the user decides that the second page in a document should not have a unique format, the user may choose the Remove button in the Custom Page Setup dialog box. When the user chooses this button, your application needs to dispose of the format object for this page and reassociate it with the default format.

Although a user may choose to create a new format for a page using the Custom Page Setup dialog box, the user may also decide not to save this format.

For example, a user may click on page 4 of a document, choose the Custom Page Setup dialog box, and modify the scaling of this page. The user may then decide not to save the new scaling information and choose the Cancel button in the dialog box. When the user chooses the Cancel button, your application needs to dispose of the newly created format object and reassociate this page with its previously saved format object.

Saving a job object and the format objects it references is discussed in the chapter “Core Printing Features” in this book.

Sharing Formats for Document Pages

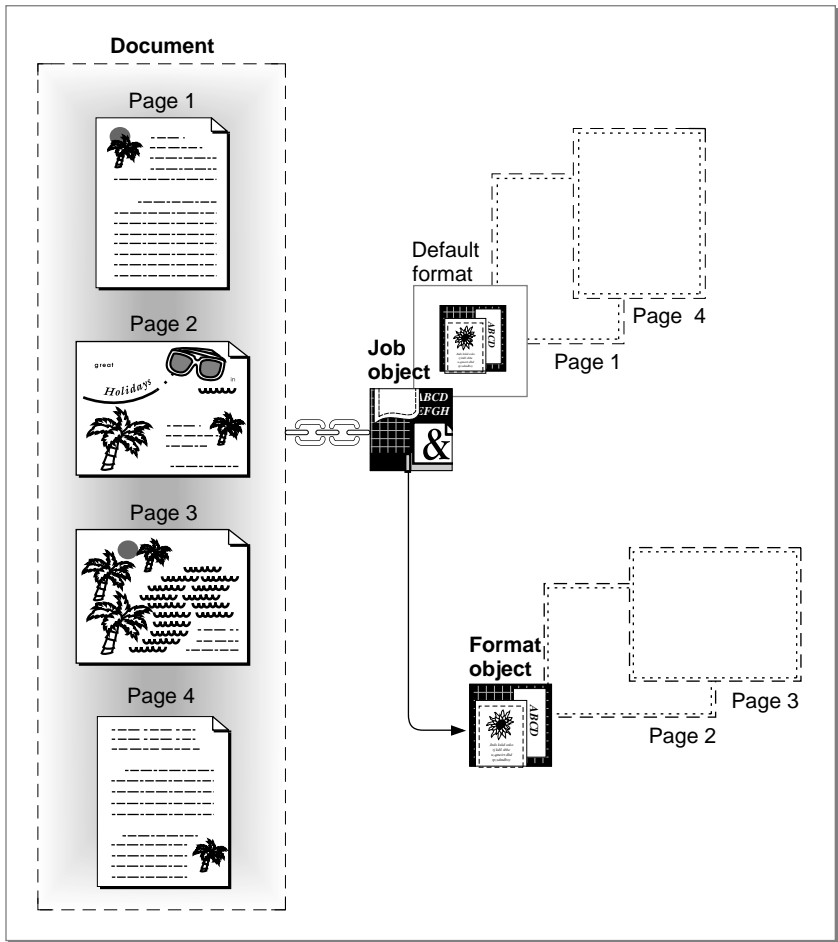
You need to clone a format object when a user wants to share a format, created using the Custom Page Setup dialog box, with an additional page in the same document. For example, a user may have a four-page document that consists of one page in landscape orientation and three pages that use the default format. A user may decide that page 3 of this document should also use landscape orientation.

When the user clicks on page 3 and chooses the Format button in the Custom Page Setup dialog box, you need to clone the format object currently used for page 2 in this document.

Page Formatting and Dialog Box Customization

Figure 3-15 shows a four-page document in which the second and third pages use the same format. Pages 1 and 4 use the default format.

Figure 3-15 A four-page document in which pages 2 and 3 use the same format object



Page Formatting and Dialog Box Customization

Listing 3-10 shows how to clone a format object when it becomes shared. You should note that the `GXCloneFormat` function increments the owner count of this format object by 1. In this example, the format object is shared by two pages in a single document, so its owner count is also 2.

Listing 3-10 Cloning a format object for two pages in a document

```
OSErr MyApplyPageFormat(MyDocumentPtr myDocument,
                        gxFormat aNewFormat)
{
    OSERR      err = noErr;
    gxFormat    pageFormat;

    /*
     * If the specified format object is not the same as the
     * default format, clone it so it can be shared by different
     * pages. If it is the default format, set the reference to
     * nil, which specifies using the default format.
     */
    if ((aNewFormat != nil) &&
        (aNewFormat != GXGetJobFormat(myDocument->documentJob, 1)))
    {
        pageFormat = GXCloneFormat(aNewFormat);
        err = GXGetJobError(myDocument->documentJob);
    }
    else
        pageFormat = nil;

    /*
     * If there are no errors, dispose of the old format object and
     * store the new one. Reformat the page, if necessary.
     */
    if (err == noErr)
    {
        if (myDocument->pageFormat[myDocument->curPage - 1] != nil)
            GXDisposeFormat
                (myDocument->pageFormat[myDocument->curPage - 1]);
        myDocument->pageFormat[myDocument->curPage - 1] = pageFormat;
    }
}
```

```
    /*  
        Place code here if your application needs to adjust the  
        document based on the new format object.  
    */  
    ...  
}  
return err;  
}
```

Disposing of a Format Object for a Page in a Document

You need to dispose of a format object when a user wants to modify a format for a single page that is also shared by other pages in the same document, the user wants to return to the default format, or the user decides not to save a format.

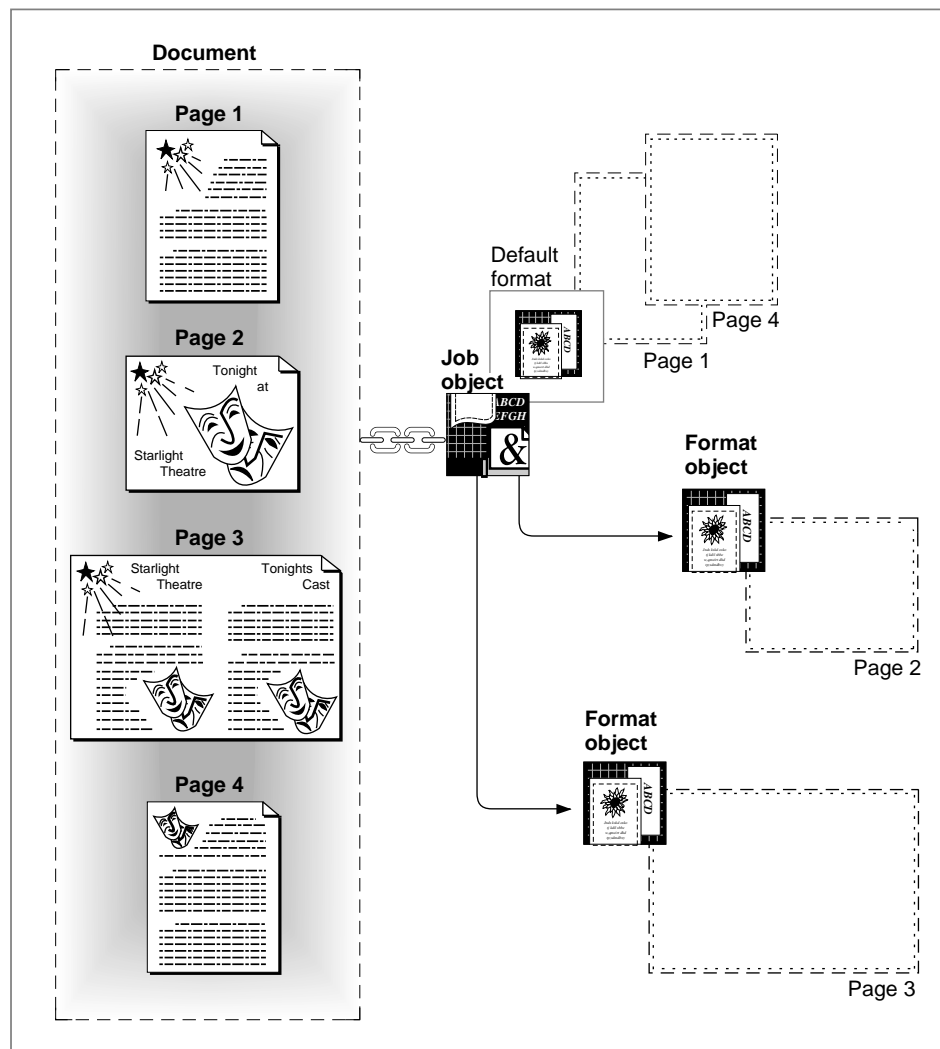
For example, a user may have a four-page document that consists of two pages in landscape orientation (pages 2 and 3) and two pages that use the default format (pages 1 and 4). A user may decide to modify the scaling of page 3 of this document. A user specifies scaling for a page in the Custom Page Setup dialog box. Note that a user also can modify scaling for the default format in the Page Setup dialog box.

When the user clicks on page 3, specifies a scaling factor, and chooses the Format button in the Custom Page Setup dialog box, you need to dispose of the format object for this page and create a new one. This user is not modifying page 2, and therefore, you should not modify or dispose of its format object.

Page Formatting and Dialog Box Customization

Figure 3-16 shows a four-page document in which the second and third pages use landscape orientation, but page 3 uses a modified scaling factor. Pages 1 and 4 use the default format.

Figure 3-16 A four-page document in which pages 2 and 3 use unique formats objects



Listing 3-11 shows how to dispose of a format object for a page in a document. In this example, you need to dispose of the format object because it is shared by another page in the document (its owner count is greater than 1).

Page Formatting and Dialog Box Customization

The `GXDisposeFormat` function decrements the owner count of this format object by 1. In this example, the format object is now used by only one page in the document, so its owner count becomes 1. Storage for a format object is removed only when its owner count becomes 0. After you call the `GXDisposeFormat` function, you need to call the `GXNewFormat` function to create a new format object for this page.

Listing 3-11 Disposing of a format object for a page in a document and creating a new one

```
OSErr MyDeletePage(MyDocumentPtr myDocument)
{
    OSErr    err;
    long     curPage, pg;

    /*
       Dispose of the current page's shape object and format
       object.
    */
    curPage = myDocument->curPage;
    GXDisposeShape(myDocument->documentPage[curPage - 1]);
    if (myDocument->pageFormat[curPage - 1] != nil)
        GXDisposeFormat(myDocument->pageFormat[curPage - 1]);

    /* Place application-specific code to delete a page here. */
    ...

    /*
       Shift all pages coming after this one to fill the gap
       created by this deletion.  When finished, decrement the
       number of pages in the document.
    */
    if (myDocument->numPages != 0)
        for (pg = curPage; pg < myDocument->numPages; pg++)
        {
            myDocument->documentPage[pg - 1] =
                                   myDocument->documentPage[pg];
            myDocument->pageFormat[pg - 1] =
                                   myDocument->pageFormat[pg];
        }
    --myDocument->numPages;
}
```

Page Formatting and Dialog Box Customization

```

/* If the current page is beyond the last page, reset it. */
if (curPage > myDocument->numPages)
    --myDocument->curPage;

/*
    Invalidate the window so that the page is updated on screen.
    Check for errors and return.
*/
InvalRect(&(myDocument->documentWindow)->portRect);
err = GXGetJobError(myDocument->documentJob);
if (err == noErr) err = (OSErr)GXGetGraphicsError(nil);
return err;
}

```

Using Forms With Format Objects

Your application may choose to support a form that can be applied to each page in a document. This may save printing time because the form can be stored in the printer's memory and need not be sent with each page of the document. For an introduction to forms, see "Forms and Format Objects," which begins on page 3-20.

To associate a form shape and its mask shape with a format object, you use the `GXSetFormatForm` function. To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function. The shape type that you associate with a format object must be a picture shape.

The `GXSetFormatForm` function replaces any form previously associated with a particular format object. It increments the owner counts of the new picture shapes (by calling the `GXCloneShape` function) and decrements the owner count of the old picture shapes (by calling the `GXDisposeShape` function).

Listing 3-12 shows how to associate a form with a format object. The `MyAddFormatForm` function in the listing adds a form consisting of a rectangle to the format object of the current page.

Listing 3-12 Adding a form to a format object

```

OSErr MyAddFormatForm(MyDocumentPtr myDocument)
{
    OSErr          err;
    long           curPage;
    gxFormat       theFormat;
    gxShape        rectShape;
    gxRectangle    pageRect;

    /*
       Get the current format object. If it's nil, use the job's
       default format object.
    */
    curPage = myDocument->curPage;
    theFormat = myDocument->pageFormat[curPage - 1];

    if (theFormat == nil)
        theFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /*
       Create a rectangle shape to use as the format object's form.
       Make the rectangle's frame the imageable area of the page.
    */
    GXGetFormatDimensions(theFormat, &pageRect, nil);
    rectShape = GXNewRectangle(&pageRect);
    GXSetShapeBounds(rectShape, &pageRect);
    GXSetShapePen(rectShape, ff(3));
    GXSetShapeFill(rectShape, gxClosedFrameFill);
    err = (OSErr) GXGetGraphicsError(nil);

    /*
       Set the format object's form to a new picture shape, check
       for errors, and then dispose of the shape.
    */
    if (err == noErr)
    {
        GXSetShapeType(rectShape, gxPictureType);
        GXSetFormatForm(theFormat, rectShape, nil);
        err = GXGetJobError(myDocument->documentJob);
    }
    GXDisposeShape(rectShape);
    return err;
}

```

Storing Halftone Information in a Format Collection

Your application can store halftone information for each page in a document in a format collection. QuickDraw GX stores the halftone structure for a format object as a collection item in the format collection. For an introduction to printing with halftones, see “Halftones and Format Collections,” which begins on page 3-21.

Halftones are described by the `gxHalftone` structure definition:

```
struct gxHalftone{
    fixed          angle;
    fixed          frequency;
    gxDotType      method;
    gxTintType     tinting;
    gxColor        dotColor;
    gxColor        backgroundColor;
    gxColorSpace   tintSpace;
};
```

The `angle` parameter describes the direction of the halftone. The `frequency` parameter describes the size of the dot, in cells per inch. The `method` parameter describes the way in which the halftone cell is filled. The `tinting` parameter describes how the desired color is converted into a ratio of color dots and background dots. The `dotColor` and `backgroundColor` parameters are the colors of the dots used to form the halftone. And the `tintSpace` parameter describes the color space that the original color is converted to before the tint value is determined. For detailed information on the `gxHalftone` structure, see the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

The `gxFormatHalftoneTag` enumerator is used to identify the `gxFormatHalftoneInfo` structure in the format collection:

```
enum { gxFormatHalftoneTag = 'half' };

struct gxFormatHalftoneInfo{
    long          numHalftones;
    gxHalftone    halftones[1];
};
```

The `numHalftones` field specifies how many `gxHalftone` entries are in the `gxFormatHalftoneInfo` structure. The `halftones` field specifies each of them.

Listing 3-13 shows how to store halftone information for a page in a format collection.

Listing 3-13 Storing halftone information in a format collection

```

OSError MySetFormatHalftones(gxFormat theFormat,
                             gxFormatHalftoneInfo *theFormatHalftones)
{
    OSError      err;
    Collection    fmtCollection;

    /*
     * Get the format collection, and attempt to delete a
     * gxFormatHalftoneTag collection item, in case one exists.
     * Then, add a new one.
     */
    fmtCollection = GXGetFormatCollection(theFormat);
    RemoveCollectionItem(fmtCollection,
                        gxFormatHalftoneTag,
                        gxPrintingTagID);
    err = AddCollectionItem(fmtCollection,
                        gxFormatHalftoneTag,
                        gxPrintingTagID,
                        sizeof(gxFormatHalftoneInfo),
                        theFormatHalftones);

    /*
     * Since we changed the format object's collection items, we
     * must call GXChangedFormat.
     */
    if (err == noErr)
        GXChangedFormat(theFormat);
    return err;
}

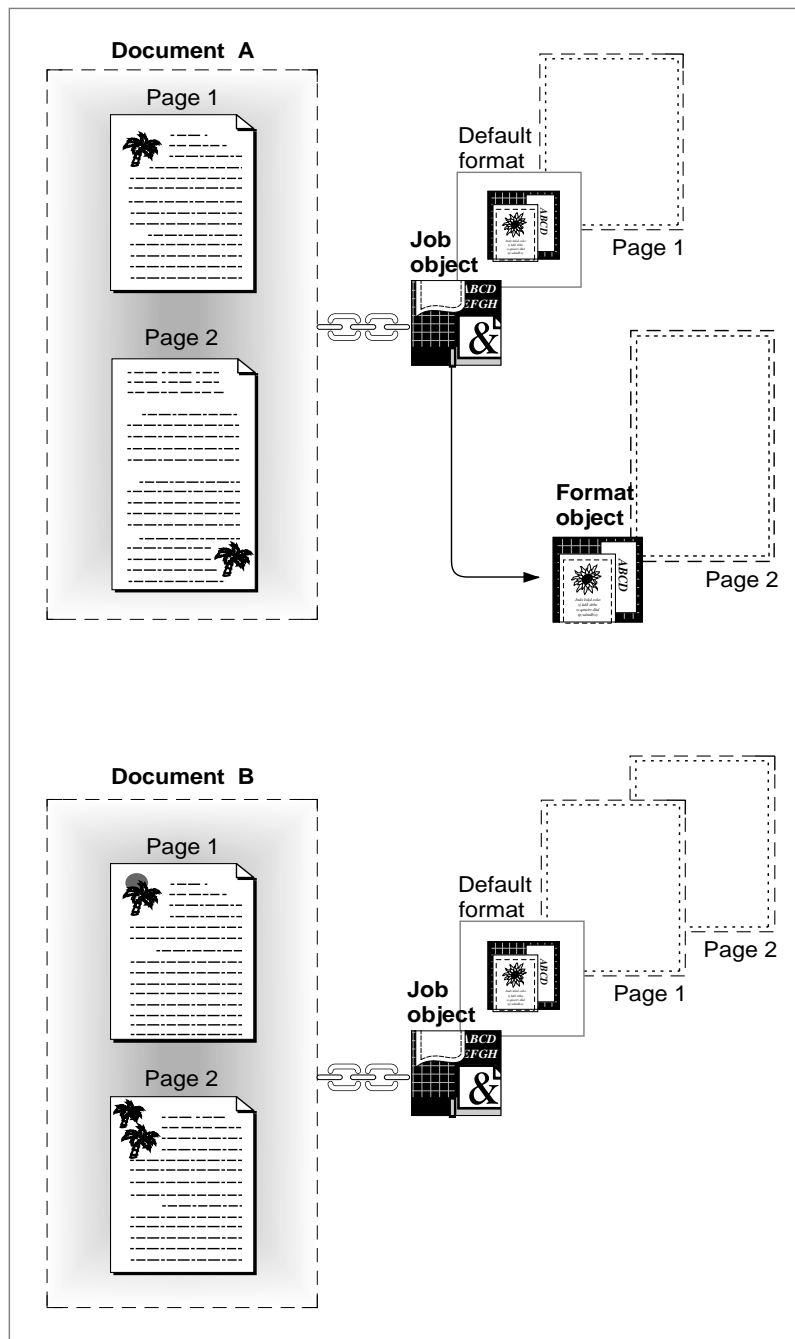
```

To provide halftone information for shape objects drawn with the same ink, you use a halftone synonym. For detailed information on how to use halftone synonyms, see the chapter “Advanced Printing Features” in this book.

Copying a Format Object for Use in Other Documents

When a user wants to disassociate a format from a particular document and associate it with another document, you use the `GXNewFormat`, `GXCopyFormat`, and `GXDisposeFormat` functions. For example, a user may have a three-page document that contains a format object for a single page in landscape orientation. This user may want to use the landscape page in another document and delete it from the original document.

Figure 3-17 shows two documents. Document A consists of two pages—one page uses the default format, the other uses a unique format object. Document B also consists of two pages—each page uses the default format. A user may decide to use the format of page 2 in Document A for page 1 of Document B.

Figure 3-17 Moving a format object from one document to another

Page Formatting and Dialog Box Customization

Listing 3-14 shows how to move a format object from one document to another. The `srcPage` and `srcDocument` parameters to the `MyMoveFormatToJob` function in the listing represent the page's location in the original document. The `destPage` and `destDocument` parameters refer to the new location and document. Initially, a format object for the destination page does not exist.

Listing 3-14 Moving a format object from one document to another

```
OSErr MyMoveFormatToJob(long srcPage, MyDocumentPtr srcDocument,
                        long destPage, MyDocumentPtr destDocument)
{
    OSErr      err;
    gxFormat    srcPgFormat, destPgFormat;

    /*
     * Get the source format object. If it is nil, create a
     * destination format object from the source job object.
     */
    srcPgFormat = srcDocument->pageFormat[srcPage-1];
    if (srcPgFormat == nil)
        srcPgFormat = GXNewFormat(srcDocument->documentJob);

    /*
     * Create a new destination format object and copy the source
     * format object to it. Then dispose of the source format
     * object and clear out the source page's reference.
     */
    destPgFormat = GXNewFormat(destDocument->documentJob);
    GXCopyFormat(srcPgFormat, destPgFormat);
    GXDisposeFormat(srcPgFormat);
    srcDocument->pageFormat[srcPage-1] = nil;

    /*
     * If there were no errors, store the destination page's format
     * object reference.
     */
    err = GXGetJobError(srcDocument->documentJob);
    if (err == noErr)
        err = GXGetJobError(destDocument->documentJob);
    if (err == noErr)
        destDocument->pageFormat[destPage-1] = destPgFormat;
    return err;
}
```

Obtaining the Mapping From a Format Object

To access a format object's mapping, your application uses the `GXGetFormatMapping` function. Listing 3-15 shows how to obtain the mapping for the format object associated with the `whichPage` page.

Listing 3-15 Obtaining a format object's mapping

```
OSErr MyGetFormatMapping(MyDocumentPtr myDocument, long whichPage,
                        gxMapping *theMapping)
{
    gxFormat    pgFormat;

    /*
       Get the current page's format. A nil reference specifies
       using the job's format object.
    */
    pgFormat = myDocument->pageFormat[whichPage - 1];
    if (pgFormat == nil)
        pgFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /* Get the format's mapping. */
    GXGetFormatMapping(pgFormat, theMapping);
    return GXGetJobError(myDocument->documentJob);
}
```

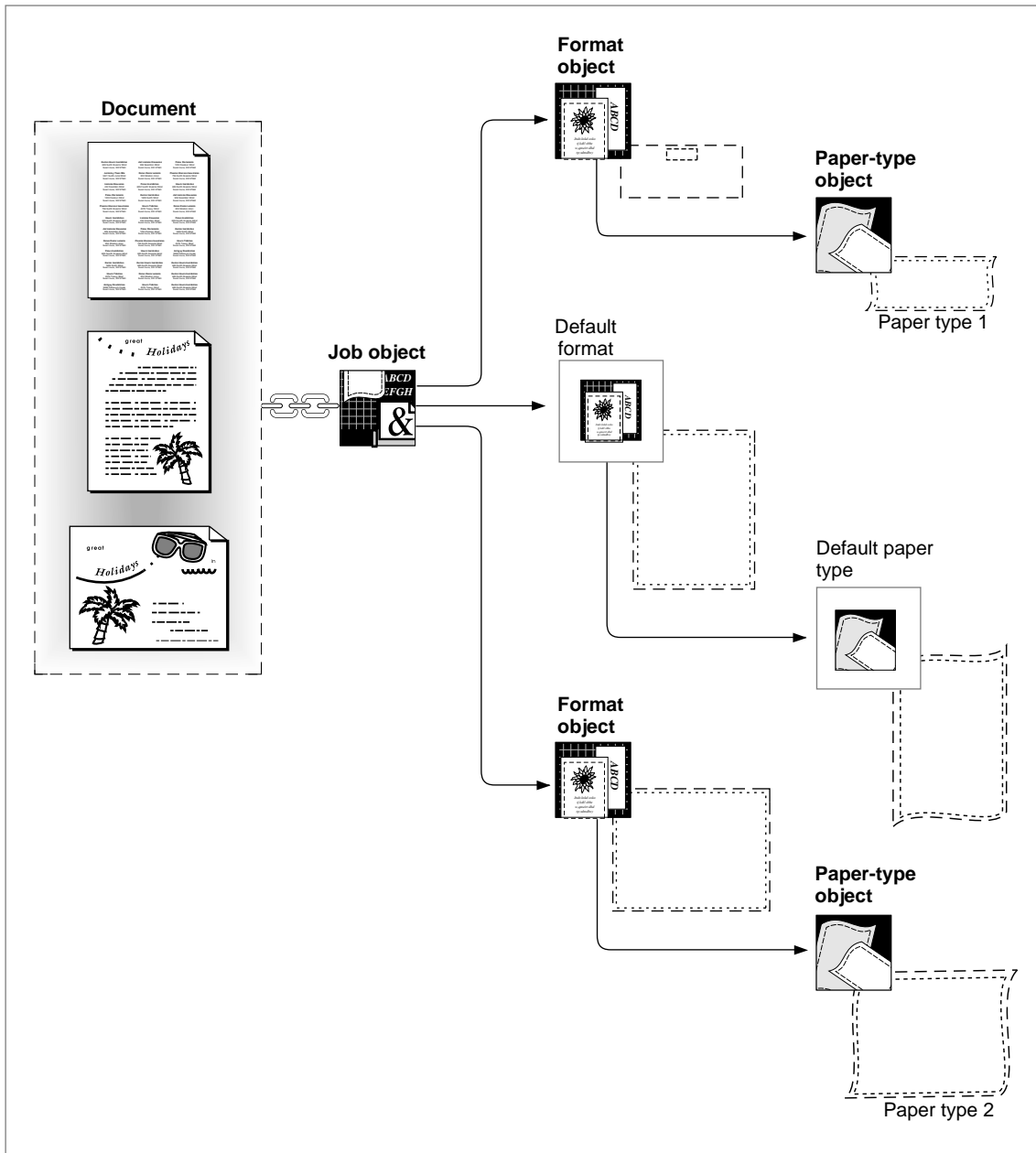
For an introduction to mapping, see “Mapping for Format Objects” beginning on page 3-18.

Obtaining a Paper-Type Object Associated With a Format

QuickDraw GX allows a user to specify a paper-type name for each page of a document. Pages with different imageable areas require different format objects. Imageable areas differ both because of physical characteristics (paper size and page size) and because of rendering characteristics (such as scaling and orientation).

Pages require different paper-type objects only when the physical characteristics differ. A change in the paper-type object requires a change in the format object. The job object in Figure 3-18 references three format objects and three paper-type objects. This allows a user to print the address page on an envelope, a letter that contains graphics on an 8.5-by-11 inch sheet of paper in portrait orientation, and a page of graphics on a sheet of paper in landscape orientation.

Figure 3-18 A three-page document and its corresponding job object, format objects, and paper-type objects



You can use the `GXGetFormatPaperType` function to obtain a format object's associated paper-type object. For detailed information on working with paper-type objects, see the chapter "Advanced Printing Features" in this book.

Listing 3-16 shows how to obtain the paper-type object that a format object references. The `MyGetPaperTypeName` function in the listing returns the name stored in the paper-type object.

Listing 3-16 Obtaining the paper-type object associated with a format object

```
OSErr MyGetPaperTypeName(MyDocumentPtr myDocument, Str255
                        paperTypeName)
{
    gxPaperType    thePaperType;
    long           curPage;
    gxFormat       pgFormat;

    /*
       Get the current page's format. A nil reference specifies
       using the job's format object.
    */
    curPage = myDocument->curPage;
    pgFormat = myDocument->pageFormat[curPage - 1];
    if (pgFormat == nil)
        pgFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /* Get the format's object paper-type object and name. */
    thePaperType = GXGetFormatPaperType(pgFormat);
    GXGetPaperTypeName(thePaperType, paperTypeName);
    return GXGetJobError(myDocument->documentJob);
}
```

Scanning Through a Job's Format Objects

QuickDraw GX allows you to scan through the format objects associated with a particular job. You can use the `GXCountJobFormats` function to obtain the number of format objects in a particular job object. If you want to examine or manipulate each format object for a job, you can use the `GXForEachJobFormatDo` function.

Note

You cannot use the `GXForEachJobFormatDo` function to modify the default format. ♦

Page Formatting and Dialog Box Customization

Listing 3-17 shows the `GXForEachJobFormatDo` function being called to execute the `MyCheckMappingProc` function on each format object.

Listing 3-17 Using the `GXForEachJobFormatDo` function

```
OSErr MyCheckAllFormatMappings(MyDocumentPtr myDocument)
{
    gxMapping    theMapping;

    /* Loop through each format, and check its mapping. */

    GXForEachJobFormatDo(myDocument->documentJob,
                        MyCheckMappingProc, (void *) &theMapping);

    return GXGetJobError(myDocument->documentJob);
}
```

The `GXForEachJobFormatDo` function passes a pointer to the application-supplied function to execute and a pointer to the information that the application-supplied function returns. The prototype for the application-supplied function is as follows:

```
gxLoopStatus MyFormatFunction (gxFormat aFormat, void *refCon);
```

The first parameter, `aFormat`, is a reference to a format object. QuickDraw GX sets this parameter as it calls the function for each format object referenced by a job object. The second parameter, `refCon`, is a pointer to a reference constant through which data can be passed. The return value, `gxLoopStatus`, specifies whether the application-supplied function should be called again, allowing you to terminate the `GXForEachJobFormatDo` function early.

Listing 3-18 shows the application-supplied function, `MyCheckMappingProc`, that obtains scaling and orientation information for each format object associated with a particular job. For example, you can use this function to obtain scaling information when you need to adjust a ruler.

Listing 3-18 Obtaining scaling information on each format object

```

pascal gxLoopStatus MyCheckMappingProc(gxFormat aFormat, void
                                         *theMapping)
{
    /*
       Get the mapping for the current format object, check it out,
       and keep looping until all formats objects are accessed.
    */
    GXGetFormatMapping(aFormat, (gxMapping *) theMapping);

    /*
       Your application could adjust rulers here, or do some
       other useful thing based on each format object's mapping.
    */
    ...
    return gxKeepLooping;
}

```

Note

For information about the `gxMapping` structure that contains scaling and rotation (orientation) information, see the mathematical functions chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. ♦

Associating Format Objects With Document Pages

Your application is responsible for managing the correspondence between format objects and individual pages in a document. For example, a user may create a document that consists of three pages. Through the Custom Page Setup dialog box, you can allow a user to specify that pages 1 and 2 use portrait orientation and page 3 uses landscape orientation. In this example, you need to store information that pages 1 and 2 use the default format, while page 3 uses a unique format object.

When a user saves a document containing multiple format objects, you need to save format collection information and then flatten the document's job object. In addition, when a user opens a document containing multiple format objects, you need to unflatten its corresponding job object and retrieve the format collection information. Flattening and unflattening a document's corresponding job object is discussed in the chapter "Core Printing Features" in this book.

Page Formatting and Dialog Box Customization

There are several methods you can use to store formatting information. A common method, shown in this section, is to save the correspondence between format objects and pages in the format collection. Listing 3-19 shows a function that performs this task.

Listing 3-19 Saving the correspondence between format objects and document pages in a format collection

```
OSErr MySaveFormatRefs(MyDocumentPtr myDocument)
{
    OSERR      err = noErr;
    Handle      theFormatIdxList;
    Collection  fmtCollection;
    gxFormat    defaultFmt;

    /* Create a handle containing all of the format object indices. */
    if (myDocument->numPages > 0)
    {

        /*
         Obtain the format collection. If you have already have a
         document page-to-format object correspondence item stored,
         remove it.
        */
        defaultFmt = GXGetJobFormat(myDocument->documentJob, 1);
        fmtCollection = GXGetFormatCollection(defaultFmt);

        if (fmtCollection != nil)
            RemoveCollectionItem(fmtCollection, kMyFormatInfoType,
                                printingTagID);

        /*
         Create a list of document page-to-format object
         correspondences for the current document. If there are no
         errors, add the item to the format collection for later
         retrieval.
        */
        err = MyCreateFormatIndexList(myDocument,
                                     &theFormatIdxList);
    }
}
```

```

    if (err == noErr)
    {
        HLock(theFormatIdxList);
        err = AddCollectionItem(fmtCollection, kMyFormatInfoType,
                                printingTagID,
                                GetHandleSize(theFormatIdxList),
                                *theFormatIdxList);
        DisposHandle(theFormatIdxList);
    }
}
return err;
}

```

The `MyCreateFormatIndexList` function stores the index of each page's format object in a handle. The index of the format object for page 1 goes in the first long word of the `theFormatIdxList` handle. The index of the next page's format object goes in the next long word, and so on. The handle is created and returned to the caller. Listing 3-20 shows the `MyCreateFormatIndexList` function.

Listing 3-20 Filling the handle

```

OSErr MyCreateFormatIndexList(MyDocumentPtr myDocument, Handle
                                *theFormatIdxList)
{
    OSErr      err;
    long       fmtIdx, pg, *idxList;
    gxFormat   curFormat;

    /*
     * Create a handle large enough to hold all of our entries. This
     * example uses NewHandleClear so that all of our indices are
     * initialized to 0 (an invalid format index). This application
     * stores a nil format reference for each page which uses the
     * default format. This allows us to indicate these "nil
     * references" by an index of 0 in our resource.
     */
    *theFormatIdxList = NewHandleClear(sizeof(long) *
                                        (myDocument->numPages));

    err = MemError();
}

```

Page Formatting and Dialog Box Customization

```

/*
    If there aren't any errors, go through each format object. If
    the format object is used by any pages of the document, store
    the format object's index in those page entries of
    theFormatIdxList. Skip format object #1, because that's the
    default format.
*/
if (err == noErr)
{
    HLock(*theFormatIdxList);
    idxList = (long *) **theFormatIdxList;

    for (fmtIdx = 2; fmtIdx <=
        GXCountJobFormats(myDocument->documentJob); fmtIdx++)
    {
        curFormat = GXGetJobFormat(myDocument->documentJob,
            fmtIdx);

        for (pg = 1; pg <= myDocument->numPages; pg++)
            if (myDocument->pageFormat[pg - 1] == curFormat)
                idxList[pg - 1] = fmtIdx;
    }
    HUnlock(*theFormatIdxList);
}
return err;
}

```

Listing 3-21 shows how to retrieve format object correspondence from a format collection when a user opens a document containing multiple format objects. This function associates new format object references with a document, based upon the format object indices that are saved with the document. The function is called when a document is opened. The format object references are stored in the passed `MyDocumentRec` structure.

Listing 3-21 Retrieving the correspondence between document pages and format objects from a format collection

```

OSErr MyAdjustFormats(MyDocumentPtr myDocument)
{
    OSErr      err = noErr;
    Handle      theFormatIdxList = nil;
    gxFormat     theFormat, defaultFmt;
    long         pg, numPages, fmtIdx, *idxList, idx, listSize,
                attrs;
    Collection   fmtCollection;

    /*
       Get the format collection, and search for one of the
       document page-to-format object correspondence items.
    */
    defaultFmt = GXGetJobFormat(myDocument->documentJob, 1);
    fmtCollection = GXGetFormatCollection(defaultFmt);

    /*
       Load the item containing the correspondences. First,
       determine if the item exists. Next, create a handle to
       hold the item, and retrieve it. Because there is one
       long-word entry for each page of the document, determine
       the number of pages in the document.
    */
    err = GetCollectionItemInfo(fmtCollection, kMyFormatInfoType,
                                gxPrintingTagID, &idx, &listSize, &attrs);
    if (err == noErr)
        theFormatIdxList = NewHandle(listSize);
    if (theFormatIdxList != nil)
    {
        HLock(theFormatIdxList);
        err = GetCollectionItem(fmtCollection, kMyFormatInfoType,
                                gxPrintingTagID, nil, *theFormatIdxList);
        numPages = listSize / sizeof(long);
    }
}

```

Page Formatting and Dialog Box Customization

```

/*
    Loop through each saved index. In this example, the first
    index is for page 1, the second is for page 2, and so on.
    Call the GXGetJobFormat function for each saved
    index. Store the format references as they are
    processed. When finished, throw away the handle.
*/
idxList = (long *) *theFormatIdxList;
for (pg = 1; pg <= numPages; pg++)
{
    fmtIdx = idxList[pg - 1];
    if (fmtIdx != nil)
        theFormat = GXGetJobFormat(myDocument->documentJob,
                                   fmtIdx);

    else
        theFormat = nil;
    myDocument->pageFormat[pg - 1] = theFormat;
}
DisposHandle(theFormatIdxList);
}
return err;
}

```

Customizing QuickDraw GX Dialog Boxes

Your application can customize QuickDraw GX dialog boxes. To customize a QuickDraw GX dialog box, you need to take the following general steps:

1. Install a message handler to override the message that causes a QuickDraw GX print dialog box to be displayed. Your override function loads your panel.
2. Create an item list ('DITL') resource that defines the items, such as radio buttons, editable text fields, checkboxes, and pop-up menus, that you want to include in your panel. You may have to create additional resources, such as a control ('CNTL') resource for pop-up menus.
3. Create an icon resource that is displayed in the extended dialog box.
4. Create a panel (gxPrintPanelType) resource that provides a name for your panel and associates the panel with the item list resource and the icon resource.
5. Install a handler to respond to events while the panel is active. This handler can be an extended item list (gxExtendedDITLType) resource or may be an override of the gxHandlePanelEvent message.

These steps need not be done in order; however, all must be completed. The following sections describe how your application adds a panel to a QuickDraw GX dialog box and how to automate the response to user actions using the extended item list (`gxExtendedDITLType`) resource.

Adding Panels to Dialog Boxes

To add a panel to a dialog box, you call the `GXInstallApplicationOverride` function to override the messages that QuickDraw GX sends to display its dialog boxes. The following call to `GXInstallApplicationOverride` sets up the `MyFormatDialogOverride` function to be called when the application receives the `gxFormatDialog` message:

```
GXInstallApplicationOverride(myDocument->documentJob,
                             gxFormatDialog, MyFormatDialogOverride);
```

The `MyFormatDialogOverride` function that is called in response to the message is as follows:

```
OSErr MyFormatDialogOverride(gxFormat aFormat, StringPtr title,
                             gxDialogResult *result)
{
    OSErr    err = noErr;

    err = MySetUpByPagePanel(aFormat, GXGetMessageHandlerResFile());
    if (!err) err = Forward_FormatDialog(aFormat, title, result);
    return err;
}
```

Note

To remove the application override when a change to the default behavior associated with the message is no longer desired, use the `GXInstallApplicationOverride` function with the function pointer set to nil to take the override out of the message chain. ♦

Because you have specified a function pointer in the `GXInstallApplicationOverride` function to override the message that displays the dialog box, QuickDraw GX calls the `MyFormatDialogOverride` function just before it displays the Custom Page Setup dialog box. The `MyFormatDialogOverride` function calls the `GXSetupDialogPanel` function for each panel that you want to add. The `MyFormatDialogOverride` function must forward the message to the next handler in the message chain.

Page Formatting and Dialog Box Customization

Listing 3-22 shows the `MySetUpByPagePanel` function, which obtains information from the collection to set up a new panel, calls `GXSetupDialogPanel`, and forwards the message.

Listing 3-22 Setting up a new panel

```
#define kCreator                'Ex#9'        /* registered
                                           application
                                           creator */

#define kMyKindaCollectionType  kCreator      /* collection
                                           tag type */

#define r_MyFormatPanelResID    6000         /* ID of the panel
                                           and panel icon
                                           resources */

typedef struct MyKindaCollectionRec {
    unsigned char  isEnabled;                /* Enabled? */
    char           fillByte;                 /* C adds this (if
                                           you don't) for
                                           alignment */
} MyKindaCollectionRec, *MyKindaCollectionPtr,
                        **MyKindaCollectionHdl;

...

OSErr MySetUpByPagePanel(gxFormat aFormat, short ourResFile)
{
    OSErr          err;
    Collection      fmtCollection;
    gxPanelSetupRecord  panelInfo;
    MyKindaCollectionRec  mySettings;

    /*
       Access the format collection and search for the collection
       object item in which the default settings are stored.
    */
    fmtCollection = GXGetFormatCollection(aFormat);

    err = GetCollectionItem(fmtCollection, kMyKindaCollectionType,
                           gxPrintingTagID, nil, &mySettings);
```



```

/*
    If the collection object item does not exist, create one and
    add it to the format collection to support default settings
    for the dialog panel.
*/
if (err == collectionItemNotFoundErr)
{
    mySettings.isEnabled = false;

    err = AddCollectionItem(fmtCollection,
                           kMyKindaCollectionType,
                           gxPrintingTagID,
                           sizeof(MyKindaCollectionRec),
                           &mySettings);
}

/*
    Install the panel. Specify its type, resource ID, and the
    resource file in which it is located.
*/
if (!err)
{
    panelInfo.panelKind      = gxApplicationPanel;
    panelInfo.panelResId     = r_MyFormatPanelResID;
    panelInfo.resourceRefNum = ourResFile;
    panelInfo.refCon         = 0; /* not being used here */
    err = GXSetupDialogPanel(&panelInfo);
}
return err;
}

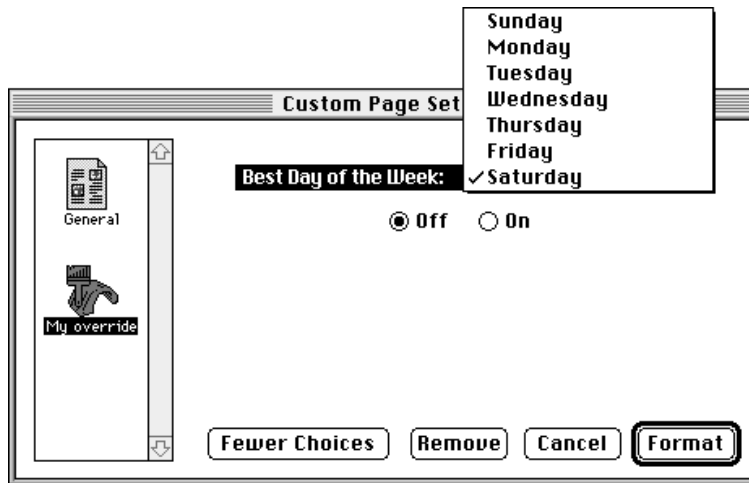
```

Once the user confirms or cancels the dialog box, QuickDraw GX disposes of all panel information. Note that while QuickDraw GX uses a resource file number supplied by the panel owner to look for panel resources, it does not leave the resource chain set to this file. The resource chain's current file is restored once the resources are retrieved.

Setting Up Dialog Box Resources

Figure 3-19 shows the panel that is loaded in Listing 3-22. Listing 3-23 through Listing 3-27 show the resources required to add this panel.

Figure 3-19 A panel added to the Custom Page Setup dialog box



Listing 3-23 shows the panel resource, which is added to the dialog box by the `MySetUpByPagePanel` function shown in Listing 3-22.

Listing 3-23 Sample panel resource

```
#define r_dayPopUpCtl    150    /* ID of the panel's pop-up CNTL */
#define r_dayPopUpMenu  160    /* ID of the panel's pop-up menu */

/* Description of panel added to dialog box. */

resource gxPrintPanelType (r_MyFormatPanelResID, sysheap,
                          purgeable)
{
    "My override", smRoman, r_MyFormatPanelResID, /* Icon ResID */
    r_MyFormatPanelResID /* Panel ResID */
};
```

Listing 3-24 shows the item list resource, 'DITL', that defines the contents of the panel.

Listing 3-24 Sample item list resource

```
resource 'DITL' (r_MyFormatPanelResID, sysheap, purgeable) {
    {
        {42, 120, 60, 166},
        RadioButton {
            enabled,
            "Off"
        },
        {42, 175, 60, 220},
        RadioButton {
            enabled,
            "On"
        },
        {14, 27, 35, 323},          /* represents the days of the week
                                   pop-up menu */
        Control {
            enabled,
            r_dayPopUpCtl
        }
    }
};
```

Note

When you design your 'DITL' resources, note that (0.0, 0.0) for a panel is at the top-left corner of the panel and not at the top-left corner of the dialog box. When you want to locate the position of the cursor within a panel, you use the `GXGetJobPanelDimensions` function to obtain the dimensions of a panel. ♦

Listing 3-25 shows the control resource, 'CNTL', that defines the pop-up menu control.

Listing 3-25 Sample 'CNTL' resource

```
resource 'CNTL' (r_dayPopUpCtl, sysheap, purgeable)
{
    {72, 4, 93, 300},
    popupTitleLeftJust,      /* menu's title is left justified */
    visible,                  /* show it */
    140,                      /* width of the menu title */
    r_dayPopUpMenu,          /* resource ID of the associated
                             menu */
    popupMenuCDEFproc + popupFixedWidth, /* type of pop-up
                                         menu */
    0,                        /* reference constant */
    "Best Day of the Week:"   /* control's title */
};
```

Listing 3-26 shows the extended item list resource that specifies how to process the items in the panel. For more information about extended item list resources, see "Automating Panel Events" beginning on page 3-25.

Listing 3-26 Sample extended item list resource

```
#define kCreator                'Ex#9'
...
#define kMyKindaCollectionType kCreator
#define kMyKindaCollectionTagID gxPrintingTagID +1
...
resource gxExtendedDITLType (r_MyFormatPanelResID,
                             sysheap, purgeable)
{
    {
        RadioButtons {kMyKindaCollectionType,
                      kMyKindaCollectionTagID, 0, {1,2}},
        PopUp {kMyKindaCollectionType,
               kMyKindaCollectionTagID, 2, 3}
    };
};
```

This extended item list resource handles two items, a pair of radio buttons, corresponding to the first two items in the 'DITL' resource, and a pop-up menu. All of these items are stored in one collection item, which is identified by the

kMyKindaCollectionType collection tag and the kMyKindaCollectionTagID item ID. The application creator is used for the collection type to distinguish it from collection items provided by QuickDraw GX. The collection item ID is simply derived from a base; in this case, gxPrintingTagID.

The status of the radio buttons occupy the first 2 bytes of the collection item (from offset 0). These bytes specify the status of items 1 and 2. The status of the pop-up menu is at offset 2. It specifies the status of item 3.

Listing 3-27 shows the 'MENU' resource, which specifies the entries in the pop-up menu. Note that the default entry is specified in the collection item.

Listing 3-27 Sample 'MENU' resource

```
resource 'MENU' (r_dayPopUpMenu, sysheap, purgeable) {
    r_dayPopUpMenu,
    textMenuProc,
    allEnabled,
    enabled,
    " ",
    {
        "Sunday",      noIcon, noKey, noMark, plain,
        "Monday",      noIcon, noKey, noMark, plain,
        "Tuesday",     noIcon, noKey, noMark, plain,
        "Wednesday",   noIcon, noKey, noMark, plain,
        "Thursday",    noIcon, noKey, noMark, plain,
        "Friday",      noIcon, noKey, noMark, plain,
        "Saturday",    noIcon, noKey, noMark, plain
    }
};
```

Parsing Page Ranges

You can install an override function for the gxParsePageRange message, which allows you to check the validity of page numbers that the user selects in the Print dialog box. You must override this message if you allow the user to specify application-specific page ranges, such as “Chapter 5.”

Listing 3-28 shows a function, MyPrintDialog, which is called in response to the user choosing the Print menu item from the File menu. The MyPrintDialog function installs an override for the gxParsePageRange message, sets up a default page range, and calls the GXPrintDialog function to display the dialog box with the default page range. After the pages have been printed, or if an error occurred while setting up the default page range, the override function for the gxParsePageRange message is removed.

Listing 3-28 Installing an override function for the `gxParsePageRange` message

```

OSErr MyPrintDialog(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxDialogResult  result;
    gxEditMenuRecord editMenuRec;

    /* Install an override function to parse page ranges. */
    GXInstallApplicationOverride(myDocument->documentJob,
                                gxParsePageRange,
                                MyParsePageRangeOverride);

    .
    .
    .
    err = MySetupDefaultPageRange(myDocument); /* not shown */
    nrequire(err, CouldNotConfigurePageRange);

    /*
       Display the Print dialog box. If there are no errors and
       the user selects the "OK" button, call a printing routine
       to output the pages.
    */
    result = GXJobPrintDialog(myDocument->documentJob,
                              &editMenuRec);
    err = GXGetJobError(myDocument->documentJob);
    if ((err == noErr) && (result == gxOKSelected))
        err = MyPrintDocument(myDocument); /* not shown */
    .
    .
    .

    /* Remove the parse page range override function. */
    CouldNotConfigurePageRange:
    GXInstallApplicationOverride(myDocument->documentJob,
                                gxParsePageRange, nil);
    return err;
}

```

The `MySetupDefaultPageRange` function that sets up a page range is not shown. For examples of setting up page ranges, see “Specifying Page Ranges in the Job Collection” on page 3-33. The `MyPrintDocument` function that prints pages is not shown. For information about printing pages, see the chapter “Core Printing Features” in this book.

Listing 3-29 shows the override function, `MyParsePageRangeOverride`, which calls another function, `MyPageRangeValidityCheck`, to validate the page range.

Listing 3-29 Override function for the `gxParsePageRange` message

```
OSErr MyParsePageRangeOverride(StringPtr fromString,
                               StringPtr toString, gxParsePageRangeResult *result)
{
    /*
     * Determine if the "To page" and "From page" strings are
     * valid. If not, the MyPageRangeValidityCheck routine
     * returns gxRangeBadFromValue or gxRangeBadToValue.
     * Otherwise it will return gxRangeParsed.
     */
    if (*result == gxRangeNotParsed)
        *result = MyPageRangeValidityCheck(fromString, toString);

    return noErr;
}
```

The `MyPageRangeValidityCheck` function is not shown. It returns `gxRangeParsed` if the page range is valid, otherwise it returns `gxRangeBadFromValue` if the From value is invalid or `gxRangeBadToValue` if the To value is invalid. For information about parse page range constants, see “The Panel Setup Structure” on page 3-101.

Page Formatting and Dialog Box Customization Reference

This section describes the constants, data types, functions, and resources that are specific to the page formatting and dialog box customization features of QuickDraw GX.

There are several sections that describe constants and data types. The following section, “Constants for Loop Status Information,” describes the constants that can be used when looping over printing-related objects. The section “Constants for Collection Item Categories and Tag IDs” on page 3-76 describes constants for manipulating printing-related collection objects. The section “Constants and Data Types for Job Collection Items” on page 3-78 describes constants and data types for job collections. The section “Constants and Data Types for Format Collection Items” on page 3-89 describes constants and data types for format collections. The section “Constants and Data Types for Paper-Type Collection Items” on page 3-94 describes constants and data types for paper-type collections.

The “Functions” section describes functions for creating and manipulating format objects, manipulating format object properties, displaying the Custom Page Setup dialog

box, obtaining information on a document's format objects, customizing QuickDraw GX dialog boxes, and accessing printing-related collection objects.

The "Application-Defined Functions" section describes message override functions for customizing QuickDraw GX dialog boxes and a function for looping through QuickDraw GX format objects associated with a particular job object.

The "Resources" section describes the panel and extended item list resources used to implement QuickDraw GX dialog boxes.

Constants for Loop Status Information

QuickDraw GX allows you to loop through the printing-related objects associated with another object. For example, QuickDraw GX allows you to loop through the format objects associated with a job object.

To allow you to loop through printing-related objects, QuickDraw GX defines loop status values in the loop status enumeration:

```
enum {
    gxStopLooping = false,
    gxKeepLooping = true
};

typedef Boolean gxLoopStatus;
```

Constant descriptions

<code>gxStopLooping</code>	If returned, QuickDraw GX stops looping through the specified printing-related objects.
<code>gxKeepLooping</code>	If returned, QuickDraw GX keeps looping through the specified printing-related objects.

Constants for Collection Item Categories and Tag IDs

This section describes the constants provided by QuickDraw GX to manipulate printing-related collections. You can use the collection tag category enumeration to determine collection item data to discard when a printer-driver switch occurs. You can use the collection tag ID enumeration to define collection objects for use with QuickDraw GX printing features.

Collection Item Categories

QuickDraw GX assigns collection object items to several collection item categories. QuickDraw GX tag categories are defined in the collection tag category enumeration, represented by `gxCollectionCategory`:


```
typedef short gxCollectionCategory;

enum {
    gxNoCollectionCategory          = (gxCollectionCategory) 0x0000,
    gxOutputDriverCategory          = (gxCollectionCategory) 0x0001,
    gxFormattingDriverCategory      = (gxCollectionCategory) 0x0002,
    gxDriverVolatileCategory        = (gxCollectionCategory) 0x0004,

    gxVolatileOutputDriverCategory =
        gxOutputDriverCategory + gxDriverVolatileCategory,
    gxVolatileFormattingDriverCategory =
        gxFormattingDriverCategory + gxDriverVolatileCategory
};
```

Constant descriptions

`gxNoCollectionCategory`

The item persists whether or not a printer-driver switch occurs or the collection is flattened.

`gxOutputDriverCategory`

The item is affected by a change in the output printer driver.

`gxFormattingDriverCategory`

The item is affected by a change in the formatting printer driver.

`gxDriverVolatileCategory`

The item is affected by a change in either the output printer driver or formatting printer driver. The item is purged when the collection is flattened if the `collectionPersistenceBit` is also set.

`gxVolatileOutputDriverCategory`

The item is purged if the output printer driver changes.

`gxVolatileFormattingDriverCategory`

The item is purged if the formatting printer driver changes.

Collection Tag ID

QuickDraw GX assigns its collection objects with the same 4-byte collection tag ID. The QuickDraw GX collection tag ID is defined in the collection tag ID enumeration:

```
enum { gxPrintingTagID = -28672 };
```

Collection tag IDs for QuickDraw GX collection objects are discussed in “About Collection Objects,” which begins on page 3-7.

Constants and Data Types for Job Collection Items

The sections that follow identify all of the collection items that QuickDraw GX provides for the job collection object.

Print-Job Information

The collection item ID for print-job information is defined in the following enumeration:

```
enum { gxJobTag = 'job ' };
```

QuickDraw GX stores print-job information in the `gxJobInfo` structure:

```
struct gxJobInfo {
    long    numPages;
    long    priority;
    long    timeToPrint;
    long    jobTimeout;
    long    firstPageToPrint;
    short   jobAlert;
    Str31   appName;
    Str31   documentName;
    Str31   userName;
};
```

Field descriptions

<code>numPages</code>	The total number of pages to print. The user specifies the page range to print in the Print dialog box.
<code>priority</code>	The print job's priority. Priorities for print jobs are defined in the print-job priorities enumeration. The user specifies the priority for a print job in the Print Time panel.
<code>timeToPrint</code>	The designated time to print a print job. The user specifies a designated printing time in the Print Time panel.
<code>jobTimeout</code>	The time to cancel the print job, in ticks. QuickDraw GX defines two print-job cancelation times in the print-job cancelation enumeration.
<code>firstPageToPrint</code>	The first page to begin printing.
<code>jobAlert</code>	When to alert the user about printing. QuickDraw GX defines print job alerts in the print-job alert enumeration.
<code>appName</code>	A string containing the name of the application used to create the printable document.
<code>documentName</code>	A string containing the name of the user's document.
<code>userName</code>	A string containing the name of the user associated with the printable document.

QuickDraw GX defines priorities for print jobs in the print-job priorities enumeration:

```
enum {
    gxPrintJobUrgent    = 0x00000001,
    gxPrintJobAtTime    = 0x00000002,
    gxPrintJobASAP      = 0x00000003
};
```

Constant descriptions

`gxPrintJobUrgent`

If set, QuickDraw GX designates a print job as “urgent.”

`gxPrintJobAtTime`

If set, QuickDraw GX designates the time to print a print job.

`gxPrintJobASAP`

If set, QuickDraw GX designates a print job as “as soon as possible.”

A holding bit for print-job priorities is defined in the following enumeration:

```
enum { gxPrintJobHoldingBit = 0x00001000 };
```

QuickDraw GX defines holding status for print jobs in the holding status enumeration:

```
enum {
    gxPrintJobHolding          = (gxPrintJobHoldingBit +
                                gxPrintJobASAP),
    gxPrintJobHoldingAtTime    = (gxPrintJobHoldingBit +
                                gxPrintJobAtTime),
    gxPrintJobHoldingUrgent    = (gxPrintJobHoldingBit +
                                gxPrintJobUrgent)
};
```

Constant descriptions

`gxPrintJobHolding`

If set, QuickDraw GX assigns a print job designated as “as soon as possible” to a holding status.

`gxPrintJobHoldingAtTime`

If set, QuickDraw GX assigns a print job designated to print at a specific time to a holding status.

`gxPrintJobHoldingUrgent`

If set, QuickDraw GX assigns a print job designated as “urgent” to a holding status.

Page Formatting and Dialog Box Customization

QuickDraw GX defines print job alerts in the print-job alert enumeration:

```
enum {
    gxNoPrintTimeAlert= 0,
    gxAlertBefore      = 1,
    gxAlertAfter       = 2,
    gxAlertBothTimes   = 3
};
```

Constant descriptions

gxNoPrintTimeAlert

If set, QuickDraw GX doesn't alert the user about printing.

gxAlertBefore If set, QuickDraw GX alerts the user that printing is about to begin.

gxAlertAfter If set, QuickDraw GX alerts the user that printing has finished.

gxAlertBothTimes

If set, QuickDraw GX alerts the user when printing begins and finishes.

QuickDraw GX defines two print-job cancelation times in the print-job cancelation enumeration, which you could use if the user failed to respond to a condition, such as out of paper:

```
enum {
    gxThirtySeconds    = 1800,
    gxTwoMinutes       = 7200
};
```

Constant descriptions

gxThirtySeconds

If set, QuickDraw GX cancels a print job in 30 seconds, or 1800 ticks.

gxTwoMinutes

If set, QuickDraw GX cancels a print job in 2 minutes, or 7200 ticks.

Collation Information

The collection item ID for collation information is defined in the following enumeration:

```
enum { gxCollationTag = 'sort' };
```

QuickDraw GX stores collation information in the collation information structure:

```
struct gxCollationInfo {
    Boolean collation;
};
```

Field descriptions

`collation` A Boolean value indicating whether the user wants to collate document pages when printed. When the user chooses the Collate Copies checkbox in the Print dialog box, the `collation` field contains `true`; otherwise, the field contains `false`.

Copies Information

The collection item ID for copies information is defined in the following enumeration:

```
enum { gxCopiesTag = 'copy' };
```

QuickDraw GX stores copies information in the copies information structure:

```
struct gxCopiesInfo {
    long copies;
};
```

Field descriptions

`copies` The number of copies of a document to print. A user specifies the number of copies to print in the Print dialog box.

The Print dialog box is discussed in the chapter “Core Printing Features” in this book.

Page-Range Information

The collection item ID for page-range information is defined in the following enumeration:

```
enum { gxPageRangeTag = 'rang' };
```

QuickDraw GX stores page-range information in the `gxPageRangeInfo` structure:

```
struct gxPageRangeInfo {
    gxSimplePageRangeInfo simpleRange;
    Str31 fromString;
    Str31 toString;
    long minFromPage;
    long maxToPage;
    char replaceString[1];
};
```

Field descriptions

`simpleRange` A string containing the page-range information structure.
`fromString` A string containing the beginning of a user-specified custom page range.
`toString` A string containing the end of a user-specified custom page range.

Page Formatting and Dialog Box Customization

<code>minFromPage</code>	The minimum default page range.
<code>maxToPage</code>	The maximum default page range.
<code>replaceString</code>	A string containing the user-specified page range from the Print dialog box. Initially, the string is one character long.

QuickDraw GX stores simple page-range information in the `gxSimplePageRangeInfo` structure:

```
struct gxSimplePageRangeInfo {
    char    optionChosen;
    Boolean printAll;
    long    fromPage;
    long    toPage;
};
```

Field descriptions

<code>optionChosen</code>	A character that contains the specific page-range option (either the default page range, replacement page range, or customized page range).
<code>printAll</code>	A Boolean value indicating whether the user wants to print all of the pages in a single document. When the user chooses the All radio button in the Print dialog box, the <code>printAll</code> field contains <code>true</code> ; otherwise, the field contains <code>false</code> .
<code>fromPage</code>	The first page in the page range to print. The user specifies a page range to print in the Print dialog box.
<code>toPage</code>	The last page in the page range to print. The user specifies a page range to print in the Print dialog box.

QuickDraw GX defines page-range options in the following enumeration:

```
enum {
    gxDefaultPageRange    = (char) 0,
    gxReplacePageRange    = (char) 1,
    gxCustomizePageRange  = (char) 2
};
```

Constant descriptions

<code>gxDefaultPageRange</code>	If set, QuickDraw GX uses a standard numeric page range; for example, the From field of the Print dialog box contains 1 and the To field contains 4.
<code>gxReplacePageRange</code>	If set, QuickDraw GX uses a single editable text field that specifies a page range; for example, a field with “Chapter 5” as the contents.
<code>gxCustomizePageRange</code>	If set, QuickDraw GX allows alphanumeric values for the From and To fields in the Print dialog box. You are responsible for validation of these values.

Quality Information

The collection item ID for quality information is defined in the following enumeration:

```
enum { gxQualityTag = 'qual' };
```

QuickDraw GX stores quality information in the `gxQualityInfo` structure:

```
struct gxQualityInfo {
    Boolean    disableQuality;
    short      defaultQuality;
    short      currentQuality;
    short      qualityCount;
    char        qualityNames[1];
};
```

Field descriptions

<code>disableQuality</code>	A Boolean value indicating whether to disable standard quality controls.
<code>defaultQuality</code>	The index of the string that represents the default quality.
<code>currentQuality</code>	The index of the string that represents the current quality.
<code>qualityCount</code>	The number of quality menu items displayed in the Quality pop-up menu in the Print dialog box.
<code>qualityNames</code>	A list of packed strings (1-byte string length preceding the actual string) that contain the menu item names (such as “Best”) displayed in the Quality pop-up menu in the Print dialog box.

File-Destination Information

The collection item ID for file-destination information is defined in the following enumeration:

```
enum { gxFileDestinationTag = 'dest' };
```

QuickDraw GX stores file-destination information in the file-destination information structure:

```
struct gxFileDestinationInfo {
    Boolean toFile;
};
```

Field descriptions

`toFile` A Boolean value indicating whether the user wants to print a document to a file. When the user chooses File in the Destination pop-up menu in the Print dialog box, the `toFile` field contains `true`. When the user chooses Printer, the `toFile` field contains `false`.

File-Location Information

The collection item ID for file-location information is defined in the following enumeration:

```
enum { gxFileLocationTag = 'floc' }
```

QuickDraw GX stores file-location information in the `gxFileLocationInfo` structure:

```
struct gxFileLocationInfo {
    FSSpec fileSpec;
};
```

Field descriptions

`fileSpec` A file system specification containing the location of the file in which to print the user's document.

File-Format Information

The collection item ID for file-format information is defined in the following enumeration:

```
enum { gxFileFormatTag = 'ffmt' };
```

QuickDraw GX stores file-format information in the `gxFileFormatInfo` structure:

```
struct gxFileFormatInfo {
    Str31 fileFormatName;
};
```

Field descriptions

`fileFormatName` A string containing the name of the format in which to print the user's document.

File-Fonts Information

The collection item ID for file-fonts information is defined in the following enumeration:

```
enum { gxFileFontsTag = 'incf' };
```

QuickDraw GX stores file-fonts information in the `gxFileFontsInfo` structure:

```
struct gxFileFontsInfo {
    char    includeFonts;
};
```

Field descriptions

`includeFonts` A character that specifies the level of fonts to include when a user prints to a file.

The level of fonts to include are defined by the following enumeration:

```
enum {
    gxIncludeNoFonts           = (char) 1,
    gxIncludeAllFonts          = (char) 2,
    gxIncludeNonStandardFonts = (char) 3
};
```

Constant descriptions

`gxIncludeNoFonts`
Do not include any fonts.

`gxIncludeAllFonts`
Include all fonts.

`gxIncludeNonStandardFonts`
Do not include standard fonts.

Paper-Feed Information

The collection item ID for paper-feed information is defined in the following enumeration:

```
enum { gxPaperFeedTag = 'feed' };
```

QuickDraw GX stores paper-feed information in the `gxPaperFeedInfo` structure:

```
struct gxPaperFeedInfo {
    Boolean    autoFeed;
};
```

Field descriptions

autoFeed A Boolean value indicating whether the user wants to use automatic or manual paper feed. When the user chooses the Automatic radio button in the Print dialog box, the `autoFeed` field contains `true`. When the user chooses the Manual radio button in the Print dialog box, the `autoFeed` field contains `false`.

Manual-Feed Information

The collection item ID for manual-feed information is defined in the following enumeration:

```
enum { gxManualFeedTag = 'manf' };
```

QuickDraw GX stores manual-feed information in the `gxManualFeedInfo` structure:

```
struct gxManualFeedInfo {
    long          numPaperTypeNames;
    Str31         paperTypeNames[1];
};
```

Field descriptions

numPaperTypeName

The number of paper-type objects to manually feed.

paperTypeNames

A string containing the names of paper-type objects to manually feed.

Standard Mapping Information

The collection item ID for standard mapping information is defined in the following enumeration:

```
enum { gxNormalMappingTag = 'nmap' };
```

QuickDraw GX stores standard mapping information in the `gxNormalMappingInfo` information structure:

```
struct gxNormalMappingInfo {
    Boolean       normalPaperMapping;
};
```

Field descriptions**normalPaperMapping**

A Boolean value indicating whether the user wants standard or special paper mapping to print a document. When the user chooses to print by specifying input-tray paper matching in the Paper Match panel, the `normalPaperMapping` field is `true`. When the user chooses to ignore paper matching and redirect the document, the `normalPaperMapping` field is `false`.

Special Mapping Information

The collection item ID for special mapping information is defined in the following enumeration:

```
enum { gxSpecialMappingTag = 'smap' };
```

QuickDraw GX stores special mapping information in the `gxSpecialMappingInfo` structure:

```
struct gxSpecialMappingInfo {
    char          specialMapping;
};
```

Field descriptions**specialMapping**

A character which specifies how to handle paper matching if the user chooses to ignore paper matching and redirect the document.

The following enumeration specifies the possible paper-mapping options:

```
enum {
    gxRedirectPages    = (char) 1,
    gxScalePages       = (char) 2,
    gxTilePages        = (char) 3
};
```

Constant descriptions**gxRedirectPages**

If set, QuickDraw GX crops the pages of a redirected document.

gxScalePages

If set, QuickDraw GX scales the pages of a document to fit the physical page size.

gxTilePages

If set, QuickDraw GX tiles the pages of a document.

Tray-Mapping Information

The collection item ID for tray-mapping information is defined in the following enumeration:

```
enum { gxTrayMappingTag = 'tmap' };
```

The tray-mapping information is defined in a `gxTrayMappingInfo` structure:

```
struct gxTrayMappingInfo {
    gxTrayIndex mapPaperToTray;
};
```

The tray index type is used to designate a specific paper tray on a printer:

```
typedef long gxTrayIndex;
```

Print-Panel Information

The collection item ID for print-panel information is defined in the following enumeration:

```
enum { gxPrintPanelTag = 'ppan' };
```

QuickDraw GX stores print-panel information in the `gxPrintPanelInfo` structure:

```
struct gxPrintPanelInfo {
    Str31    startPanelName;
};
```

Field descriptions

`startPanelName`

A string containing the name of the first panel to display in the Print dialog box.

Format-Panel Information

The collection item ID for format-panel information is defined in the following enumeration:

```
enum { gxFormatPanelTag = 'fpan' };
```

QuickDraw GX stores format-panel information in the `gxFormatPanelInfo` structure:

```
struct gxFormatPanelInfo {
    Str31    startPanelName;
};
```

Field descriptions**startPanelName**

A string containing the name of the first panel to display in the Page Setup dialog box.

Paper-Mapping Information

The collection item ID for paper-mapping information is defined in the following enumeration:

```
enum { gxPaperMappingTag = 'pmap' };
```

This collection item contains the flattened paper type that was selected for redirection.

Translated-Document Information

The collection item ID for translated-document information is defined in the following enumeration:

```
enum { gxTranslatedDocumentTag = 'trns' };
```

QuickDraw GX stores translated-document information in the `gxTranslatedDocumentInfo` structure:

```
struct gxTranslatedDocumentInfo {
    long translatorInfo;
};
```

Field descriptions

translatorInfo A value that specifies translation information for the document.

Constants and Data Types for Format Collection Items

The sections that follow identify all of the collection items that QuickDraw GX provides for the format collection object.

Orientation Information

The collection item ID for orientation information is defined in the following enumeration:

```
enum { gxOrientationTag = 'layo' };
```

Page Formatting and Dialog Box Customization

QuickDraw GX stores orientation information in the `gxOrientationInfo` structure:

```
struct gxOrientationInfo {
    char    orientation;
} ;
```

Field descriptions

orientation A character that contains the orientation information. For example, a user may choose to print a document in portrait, landscape, or rotated landscape orientation.

QuickDraw GX defines orientation options in the following enumeration:

```
enum {
    gxPortraitLayout          = (char) 0,
    gxLandscapeLayout         = (char) 1,
    gxRotatedPortraitLayout   = (char) 2,
    gxRotatedLandscapeLayout  = (char) 3
};
```

Constant descriptions

gxPortraitLayout If set, QuickDraw GX uses portrait orientation for the user-specified page.

gxLandscapeLayout If set, QuickDraw GX uses landscape orientation for the user-specified page.

gxRotatedPortraitLayout If set, QuickDraw GX uses rotated portrait orientation for the page.

gxRotatedLandscapeLayout If set, QuickDraw GX uses rotated landscape orientation for the user-specified page.

Scaling Information

The collection item ID for scaling information is defined in the scaling information enumeration:

```
enum { gxScalingTag = 'scal' };
```

QuickDraw GX stores scaling information in the `gxScalingInfo` structure:

```
struct gxScalingInfo{
    Fixed    horizontalScaleFactor;
    Fixed    verticalScaleFactor;
    short    minScaling;
    short    maxScaling;
};
```

Field descriptions

<code>horizontalScaleFactor</code>	The current horizontal scaling factor.
<code>verticalScaleFactor</code>	The current vertical scaling factor.
<code>minScaling</code>	The minimum current scaling factor.
<code>maxScaling</code>	The maximum current scaling factor.

Direct-Mode Information

The collection item ID for direct-mode information is defined in the following enumeration:

```
enum { gxDirectModeTag = 'dirm' };
```

QuickDraw GX stores direct-mode information in the `gxDirectModeInfo` structure:

```
struct gxDirectModeInfo {
    Boolean    directModeOn;
};
```

Field descriptions

<code>directModeOn</code>	A Boolean value indicating whether the user wants to print using direct mode. When the user chooses the Direct checkbox in the Page Setup dialog box, the <code>directModeOn</code> field contains <code>true</code> ; otherwise, this field contains <code>false</code> .
---------------------------	--

Format-Halftone Information

The collection item ID for halftone information is defined in the halftone information enumeration:

```
enum { gxFormatHalftoneTag = 'half' };
```

QuickDraw GX stores halftone information in the `gxFormatHalftoneInfo` structure:

```
struct gxFormatHalftoneInfo {
    long    numHalftones;
    gxHalftone  halftones[1];
};
```

Field descriptions

`numHalftones` The number of halftones in the structure.

`halftones` The halftones to use when rendering a page with this format.

Page-Inversion Information

The collection item ID for page-inversion information is defined in the following enumeration:

```
enum { gxInvertPageTag = 'invp' };
```

QuickDraw GX stores page-inversion information in the `gxInvertPageInfo` structure:

```
struct gxInvertPageInfo {
    Boolean  invert;
};
```

Field descriptions

`invert` The user-specified page-inversion information, which indicates whether a user chooses to invert a page before printing. If `true`, the page is inverted; otherwise, it is not inverted.

Horizontal Page-Flip Information

The collection item ID for horizontal page-flip information is defined in the following enumeration:

```
enum { gxFlipPageHorizontalTag = 'flph' };
```

QuickDraw GX stores horizontal page-flip information in the `gxFlipPageHorizontalInfo` structure:

```
struct gxFlipPageHorizontalInfo {
    Boolean flipHorizontal;
};
```

Field descriptions

`flipHorizontal`

The user-specified horizontal page-flip information. If `true`, a user chooses to horizontally flip the x coordinate on a page before printing.

Vertical Page-Flip Information

The collection item ID for vertical page-flip information is defined in the following enumeration:

```
enum { gxFlipPageVerticalTag = 'flpv' };
```

QuickDraw GX stores vertical page-flip information in the `gxFlipPageVerticalInfo` structure:

```
struct gxFlipPageVerticalInfo {
    Boolean flipVertical;
};
```

Field descriptions

`flipVertical`

The user-specified vertical page-flip information. If `true`, a user chooses to vertically flip the y coordinate on a page before printing.

Precise-Bitmap Information

The collection item ID for precise-bitmap information is defined in the following enumeration:

```
enum { gxPreciseBitmapsTag = 'pbmp' };
```

Page Formatting and Dialog Box Customization

QuickDraw GX stores page bitmap information in the `gxPreciseBitmapInfo` structure:

```
struct gxPreciseBitmapInfo {
    Boolean  preciseBitmaps;
};
```

Field descriptions

`preciseBitmaps`

The user-specified precise-bitmap information. If `true`, a user chooses to scale a page by 96%.

Paper-Type Lock Information

The collection item ID for lock information is defined in the following enumeration:

```
enum { gxPaperTypeLockTag = 'ptlk' };
```

QuickDraw GX stores paper-type object lock information in the `gxPaperTypeLockInfo` structure:

```
struct gxPaperTypeLockInfo {
    Boolean  paperTypeLocked;
};
```

Field descriptions

`paperTypeLocked`

A Boolean value indicating whether a paper-type object is locked.

Constants and Data Types for Paper-Type Collection Items

The sections that follow identify all of the collection items QuickDraw GX provides for the paper-type collection object.

Base Information

The collection item ID for base information is defined in the following enumeration:

```
enum { gxBaseTag = 'base' };
```

QuickDraw GX stores base information in the `gxBaseInfo` structure:

```
struct gxBaseInfo {
    long  baseType;
};
```

Field descriptions

baseType The user-specified base information, which indicates whether the source of the paper is unknown, US Letter, US Legal, A4, B5, or tabloid.

QuickDraw GX defines paper-type object base types in the following enumeration:

```
enum {
    gxUnknownBase    = 0,
    gxUsLetterBase   = 1,
    gxUsLegalBase    = 2,
    gxA4LetterBase   = 3,
    gxB5LetterBase   = 4,
    gxTabloidBase    = 5
};
```

Constant descriptions

gxUnknownBase If set, the base type is unknown.

gxUsLetterBase If set, QuickDraw GX uses a US Letter base type.

gxUsLegalBase If set, QuickDraw GX uses a US Legal base type.

gxA4LetterBase If set, QuickDraw GX uses an A4 base type.

gxB5LetterBase If set, QuickDraw GX uses a B5 base type.

gxTabloidBase If set, QuickDraw GX uses a tabloid base type.

Creator Information

The collection item ID for creator information is defined in the following enumeration:

```
enum { gxCreatorTag = 'crea' };
```

QuickDraw GX stores paper-type object creator information in the `gxCreatorInfo` structure:

```
struct gxCreatorInfo {
    OSType    creator;
};
```

Field descriptions

creator An operating-system type that contains the creator type of a paper-type object. You specify a system paper-type object creator as 'sypt', and you specify a user paper-type object creator as 'uspt'.

Page Formatting and Dialog Box Customization

Applications do not need to set this collection item. Printer drivers that create paper-type objects should use the creator type that identifies the printer driver. For example, a printer driver for the LaserWriter SC should specify a paper-type object creator as 'lwsc'.

QuickDraw GX defines paper-type object creator types in the following enumeration:

```
enum {
    gxSysPaperType    = 'sypt',
    gxUserPaperType   = 'uspt'
};
```

Constant descriptions

gxSysPaperType If set, QuickDraw GX uses a system-defined paper-type object.

gxUserPaperType If set, QuickDraw GX uses a user-defined paper-type object.

Units Information

The collection item ID for units information is defined in the following enumeration:

```
enum { gxUnitsTag = 'unit' };
```

QuickDraw GX stores units information in the `gxUnitsInfo` structure:

```
struct gxUnitsInfo {
    char    units;
} ;
```

Field descriptions

units A character that contains the units for a paper-type object. Units can be specified in picas, millimeters, and inches.

QuickDraw GX defines paper-type object units in the following enumeration:

```
enum {
    gxPicas = (char) 0,
    gxMms   = (char) 1,
    gxInches = (char) 2
};
```

Constant descriptions

gxPicas If set, QuickDraw GX uses picas to define paper-type object units.

gxMms If set, QuickDraw GX uses millimeters to define paper-type object units.

gxInches If set, QuickDraw GX uses inches to define paper-type object units.

Flags Information

The collection item ID for flags information is defined in the following enumeration:

```
enum { gxFlagsTag = 'flag' };
```

QuickDraw GX stores flags information in the following structure:

```
struct gxFlagsInfo{
    long flags;
};
```

Field descriptions

flags The flags information for a paper-type object. A flag is a bit position that indicates the system software version used to create a paper-type object.

QuickDraw GX defines paper-type object flags in the following enumeration:

```
enum {
    gxOldPaperTypeFlag      = 0x00800000,
    gxNewPaperTypeFlag      = 0x00400000,
    gxOldAndNewPaperTypeFlag= 0x00C00000,
    gxDefaultPaperTypeFlag  = 0x00100000,
};
```

Constant descriptions

gxOldPaperTypeFlag A paper type used only with applications that do not support QuickDraw GX printing.

gxNewPaperTypeFlag A paper type used only with applications that do support QuickDraw GX printing.

gxOldAndNewPaperTypeFlag A paper type used with applications that support QuickDraw GX printing and with those that do not.

gxDefaultPaperTypeFlag The default paper type.

Comment Information

The collection item ID for comment information is defined in the following enumeration:

```
enum { gxCommentTag = 'cmnt' };
```

QuickDraw GX stores comment information in the `gxCommentInfo` structure:

```
struct gxCommentInfo {
    Str255    comment;
};
```

Field descriptions

<code>comment</code>	A string containing an application-specified comment to associate with a paper-type object.
----------------------	---

Panel-Related Constants and Data Types

The following sections describe the constants and data types related to panels.

The Panel Information Structure

The panel information structure, of data type `gxPanelInfoRecord`, provides information to the panel about the current dialog box and panel event. This structure is used with the `GXHandlePanelEvent` and `GXFilterPanelEvent` override functions, whose descriptions begin on page 3-123.

```
struct gxPanelInfoRecord {
    gxPanelEvent    panelEvt;
    short           panelResId;
    DialogPtr       pDlg;
    EventRecord      *theEvent;
    short           itemHit;
    short           itemCount;
    short           evtAction;
    short           errorStringId;
    gxFormat         theFormat;
    void            *refCon;
};
```

Field descriptions

<code>panelEvt</code>	The event to filter or handle.
<code>panelResId</code>	The resource ID of the current panel (<code>gxPrintPanelType</code>) resource.
<code>pDlg</code>	A pointer to the dialog box structure.
<code>theEvent</code>	A pointer to the event that occurred.
<code>itemHit</code>	The actual item number where the event occurred, using the item-numbering scheme of the Dialog Manager.
<code>itemCount</code>	The item count before your panel's items in the dialog box.

evtAction	The action that results once this event is processed. This value is one of the constants defined in the panel event actions enumeration, which is described on page 3-101. This field is only meaningful for filtering, and is used for parsing.
errorStringId	The ID of the 'STR' resource to put in the error alert. A value of 0 in this field indicates that there is no error string to display.
theFormat	The current format. This is only meaningful in a Custom Page Setup dialog box.
refCon	A reference constant for use by the generator of the panel.

Panel Events

The panel event enumeration defines the possible event types that can occur in a panel. This data type is used with the panel information structure, which is described in the previous section.

```
enum {
    gxPanelNoEvt          = (gxPanelEvent) 0,
    gxPanelOpenEvt        = (gxPanelEvent) 1,
    gxPanelCloseEvt       = (gxPanelEvent) 2,
    gxPanelHitEvt         = (gxPanelEvent) 3,
    gxPanelActivateEvt    = (gxPanelEvent) 4,
    gxPanelDeactivateEvt  = (gxPanelEvent) 5,
    gxPanelIconFocusEvt   = (gxPanelEvent) 6,
    gxPanelPanelFocusEvt  = (gxPanelEvent) 7,
    gxPanelFilterEvt      = (gxPanelEvent) 8,
    gxPanelCancelEvt      = (gxPanelEvent) 9,
    gxPanelConfirmEvt     = (gxPanelEvent) 10,
    gxPanelDialogEvt      = (gxPanelEvent) 11,
    gxPanelOtherEvt       = (gxPanelEvent) 12,
    gxUserWillConfirmEvt  = (gxPanelEvent) 13
};
```

```
typedef long gxPanelEvent;
```

Constant descriptions

gxPanelNoEvt	No event has occurred.
gxPanelOpenEvt	The panel is about to open. It needs to be initialized and drawn.
gxPanelCloseEvt	The panel is about to close.
gxPanelHitEvt	The user has selected an item in the panel.
gxPanelActivateEvt	The dialog box in which the panel resides has just been activated.

Page Formatting and Dialog Box Customization

<code>gxPanelDeactivateEvt</code>	The dialog box in which the panel resides is about to be deactivated.
<code>gxPanelIconFocusEvt</code>	The focus has changed from the panel to the icon list.
<code>gxPanelPanelFocusEvt</code>	The focus has changed from the icon list to the panel.
<code>gxPanelFilterEvt</code>	The panel event needs to be filtered.
<code>gxPanelCancelEvt</code>	The user has selected the Cancel button in the dialog box.
<code>gxPanelConfirmEvt</code>	The user has selected the OK button in the dialog box.
<code>gxPanelDialogEvt</code>	An event has occurred in the panel that is going to be handled by a dialog box handler such as the application, a printing extension, a printer driver, or the Macintosh system software.
<code>gxPanelOtherEvt</code>	A different kind of event, such as an operating-system event, has occurred in the panel.
<code>gxPanelUserWillConfirmEvt</code>	The user has selected the confirm button, which means that it is time to parse panel interdependencies.

Panel Responses

A handler of a panel in a dialog box (including applications, printing extensions, printer drivers, and Macintosh system software) can return any value of type `OSErr` as the result of handling the panel. In addition, a panel handler can return an event of type `gxPanelResult`, as shown here. This data type is used with the `GXHandlePanelEvent` override function, which is described on page 3-123.

```
enum {
    gxPanelNoResult          = 0,
    gxPanelCancelConfirmation = 1
};

typedef long gxPanelResult;
```

Constant descriptions

<code>gxPanelNoResult</code>	The result field does not currently have any meaning.
<code>gxPanelCancelConfirmation</code>	This result is only valid if the panel event (as described in the previous section) was of type <code>gxPanelUserWillConfirmEvt</code> . After the user confirms the panel, if the panel handler discovers that the user entered an inappropriate value, the panel handler alerts the user to the problem and generates this response, which tells

QuickDraw GX to not confirm the dialog box. This allows the user the opportunity to fix the problem.

Panel Event Actions

The panel event actions enumeration defines the constants used in the `evtAction` field of the panel information structure, which is described on page 3-98. Each value defines what action takes place after an event is processed.

```
enum {
    gxOtherAction          = 0,
    gxClosePanelAction     = 1,
    gxCancelDialogAction   = 2,
    gxConfirmDialogAction  = 3
};
```

Constant descriptions

<code>gxOtherAction</code>	The current item does not change after processing this event.
<code>gxClosePanelAction</code>	The panel is closed after this event is processed.
<code>gxCancelDialogAction</code>	The dialog box is canceled after this event is processed.
<code>gxConfirmDialogAction</code>	The dialog box is confirmed after this event is processed.

The Panel Setup Structure

The panel setup structure, of data type `gxPanelSetupRecord`, is passed to the `GXSetupDialogPanel` function when the user displays a dialog box.

```
struct gxPanelSetupRecord {
    gxPrintingPanelKind    panelKind;
    short                  panelResId;
    short                  resourceRefNum;
    void                   *refCon;
};
```

Field descriptions

<code>panelKind</code>	The kind of program that is using this panel. This value is one of the constants defined in the printing panel kinds enumeration, which is described in the next section.
<code>panelResId</code>	The resource ID of the panel ('ppnl ') resource for the dialog box panel.
<code>resourceRefNum</code>	The resource file reference number for the panel.
<code>refCon</code>	A reference constant for use by the creator of the panel.

Printing Panel Kinds

The printing panel kinds enumeration provides constants for use in the `panelKind` field of the panel setup structure, which is described in the previous section.

```
enum {
    gxApplicationPanel= (gxPrintingPanelKind) 0,
    gxExtensionPanel   = (gxPrintingPanelKind) 1,
    gxDriverPanel      = (gxPrintingPanelKind) 2
};
```

```
typedef long gxPrintingPanelKind;
```

Constant descriptions

`gxApplicationPanel` A panel created for an application.

`gxExtensionPanel` A panel created for a printing extension.

`gxDriverPanel` A panel created for a printer driver.

Parse Range Results

The parse range results enumeration provides the constants that are used to parse dialog box item responses.

```
enum {
    gxRangeNotParsed      = (gxParsePageRangeResult) 0,
    gxRangeParsed         = (gxParsePageRangeResult) 1,
    gxRangeBadFromValue   = (gxParsePageRangeResult) 2,
    gxRangeBadToValue     = (gxParsePageRangeResult) 3
};
```

```
typedef long gxParsePageRangeResult;
```

Constant descriptions

`gxRangeNotParsed` QuickDraw GX has not yet parsed a page range in the string.

`gxRangeParsed` QuickDraw GX has successfully parsed a page range in the string.

`gxRangeBadFromValue` QuickDraw GX has encountered an invalid value in the “from page” string during the parse.

`gxRangeBadToValue` QuickDraw GX has encountered an invalid value in the “to page” string during the parse.

Functions

This section describes the functions for creating and manipulating format objects, manipulating format object properties, displaying the Custom Page Setup dialog box, obtaining information on a document's format objects, customizing QuickDraw GX dialog boxes, and accessing printing-related collection objects.

Included with each function description is a list of specific result codes returned by QuickDraw GX. In addition to these result codes, you may also receive file-system, memory, and resource errors. For a complete listing of specific file-system, memory, and resource errors, see *Inside Macintosh: C Summary* or *Inside Macintosh: Pascal Summary*.

You should note that not all possible result codes for a particular function are included in function descriptions within this section. For example, the Message Manager, described in *Inside Macintosh: QuickDraw GX Environment and Utilities*, allows QuickDraw GX functions to send specific messages to your application. These messages can also generate errors.

IMPORTANT

All printing functions in QuickDraw GX, with the exception of the `GXGetJobError` function, may move Macintosh memory. The `GXGetJobError` function, however, relies on data that may also move. Therefore, your application should never call a QuickDraw GX printing-related function at interrupt time. ▲

Creating and Manipulating Format Objects

When a user creates a new document, clicks on a page, and chooses the Format button in the Custom Page Setup dialog box, you use the `GXNewFormat` function to create a new format object.

When a user wants to modify a format for a single page that is also shared by other pages in the same document, the user wants to return to the default format, or the user decides not to save a format, you use the `GXDisposeFormat` function to dispose of the format object, which decrements its owner count.

When a user wants to disassociate a format from a particular document and associate it with another document, you use the `GXCopyFormat` function to copy a format object. When a user wants to share a format, created using the Custom Page Setup dialog box, with an additional page in the same document, you use the `GXCloneFormat` function to clone a format object. This function increments the owner count.

You can use the `GXCountJobFormats` function to obtain the number of format objects in a particular document, and you can use the `GXForEachJobFormatDo` function to make changes to each format object associated with a printable document. You can use the `GXCountFormatOwners` function to determine the number of references to a format object.

GXNewFormat

You can use the `GXNewFormat` function to create a format object.

```
gxFormat GXNewFormat (gxJob aJob);
```

`aJob` A reference to the job object to be associated with the new format object.

function result A reference to a format object.

DESCRIPTION

The `GXNewFormat` function creates a new format object and copies the default format for the specified job object. The `GXNewFormat` function sets the owner count to 1. You need to call this function each time a user creates a new format for a page in a document.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment for QuickDraw GX printing features failed to load due to low memory or disk errors.
<code>gxPaperTypeNotFound</code>	The default paper-type object cannot be located.

SEE ALSO

Listing 3-9 on page 3-42 shows how to use the `GXNewFormat` function to create a format object.

To dispose of a format object, see the description of the `GXDisposeFormat` function in the next section.

GXDisposeFormat

You can use the `GXDisposeFormat` function to dispose of a format object.

```
void GXDisposeFormat (gxFormat aFormat);
```

`aFormat` A reference to the format object whose owner count you want to decrement.

DESCRIPTION

You use the `GXDisposeFormat` function when you no longer need the format object. The function decrements the format's owner count. When the owner count reaches 0, the format object is deleted.

SPECIAL CONSIDERATIONS

You should *not* call this function for the default format object unless you have cloned it.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

SEE ALSO

Listing 3-9 on page 3-42 shows how to use the `GXDisposeFormat` function to dispose of a format object.

GXCopyFormat

You can use the `GXCopyFormat` function to create a copy of a format object.

```
gxFormat GXCopyFormat (gxFormat srcFormat, gxFormat dstFormat);
```

`srcFormat` A reference to the source format object to copy.

`dstFormat` A reference to the destination format object.

function result A reference to a format object.

DESCRIPTION

The `GXCopyFormat` function copies the properties from the source format object into the destination object and returns a reference to the destination format object. If you specify `nil` for the `dstFormat` parameter, QuickDraw GX creates a format object to receive the properties.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

SEE ALSO

Listing 3-14 on page 3-56 shows how to use the `GXCopyFormat` function to copy a format object's storage.

GXCloneFormat

You can use the `GXCloneFormat` function to increment the owner count of a format object by 1.

```
gxFormat GXCloneFormat (gxFormat aFormat);
```

`aFormat` A reference to the format object you wish to clone.

function result A reference to a format object.

DESCRIPTION

When a user wants to share a format with another page in the same document, you use the `GXCloneFormat` function to increment the owner count of a format object by 1, which prevents it from being deleted if it is disposed of when the other page no longer needs the format.

You can use the `GXCountFormatOwners` function to obtain the current owner count of a format object.

RESULT CODES

`gxSegmentLoadFailedErr` A required code segment could not be found, or there was not enough memory to load it.

SEE ALSO

Listing 3-10 on page 3-46 shows how to use the `GXCloneFormat` function to increment the owner count of a format object by 1.

The `GXNewFormat` function, which creates a new format object with an owner count of 1, is described on page 3-104.

The `GXDisposeFormat` function for decrementing the owner count of a format object by 1 is described on page 3-104.

The `GXCountFormatOwners` function for obtaining the current owner count of a format object is described on page 3-107.

GXCountJobFormats

You can use the `GXCountJobFormats` function to obtain the number of format objects in a job object.

```
long GXCountJobFormats (gxJob aJob);
```

`aJob` A reference to the job object in which to count the format objects.

function result The number of format objects for the job object.

DESCRIPTION

The `GXCountJobFormats` function determines the number of format objects associated with a particular job object and returns 1 if the default format is the only format object associated with a job object. A job object may contain any number of format objects.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

GXCountFormatOwners

You can use the `GXCountFormatOwners` function to determine the owner count of a format object.

```
long GXCountFormatOwners (gxFormat aFormat);
```

`aFormat` A reference to the format object in which to obtain the owner count.

function result The owner count.

DESCRIPTION

The `GXCountFormatOwners` function returns the current number of references to the format object specified by the `aFormat` parameter. The `GXNewFormat` function sets the owner count to 1. The `GXCloneFormat` function increments the owner count of a format object by 1, and the `GXDisposeFormat` function decrements the owner count by 1. When the owner count reaches 0, QuickDraw GX disposes of the format object.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

SEE ALSO

The `GXNewFormat` function, which creates a new format object with an owner count of 1, is described on page 3-104.

The `GXCloneFormat` function, which increments the owner count of a format object by 1, is described on page 3-106.

The `GXDisposeFormat` function for decrementing the owner count of a format object by 1 is described on page 3-104.

GXForEachJobFormatDo

You can use the `GXForEachJobFormatDo` function to manipulate each format object in a particular job object.

```
void GXForEachJobFormatDo (gxJob aJob, gxFormatProc aFormatProc,
                          void *refCon);
```

<code>aJob</code>	A reference to the job object associated with a particular format object.
-------------------	---

<code>aFormatProc</code>	A pointer to the function to call for each format object in a job object.
--------------------------	---

<code>refCon</code>	The reference constant passed to the function.
---------------------	--

DESCRIPTION

The `GXForEachJobFormatDo` function calls the application-defined function specified in the `aFormatProc` parameter for each format object associated with the job object specified in the `aJob` parameter. The `GXForEachJobFormatDo` function terminates when the application-defined function returns `gxStopLooping` or all format objects associated with the job object have been processed. The first format object to be processed is the default format.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

SEE ALSO

Listing 3-17 on page 3-60 shows how to use the `GXForEachJobFormatDo` function to access a format object function for each format object in a particular job object.

For information about setting up the function that is called each time through the loop, see “Looping Through Format Objects” on page 3-126.

Manipulating Format Object Properties

You use the `GXGetFormatMapping` function to obtain a format object’s mapping.

You use the `GXGetFormatPaperType` function to obtain a format object’s paper-type object.

To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function. To associate a form and its mask shape with a format object, you use the `GXSetFormatForm` function.

You call the `GXChangedFormat` function each time you change a format collection associated with a format object.

GXGetFormatMapping

You can use the `GXGetFormatMapping` function to obtain the mapping for a format object.

```
void GXGetFormatMapping (gxFormat aFormat, gxMapping *aMapping);
```

`aFormat` A reference to the format object for which to obtain the mapping.

`aMapping` On return, the mapping for a format object.

function result None.

DESCRIPTION

The `GXGetFormatMapping` function returns a mapping for a format object that is a mathematical representation of the format object’s scaling and orientation settings.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

SEE ALSO

Listing 3-15 on page 3-57 shows how to use the `GXGetFormatMapping` function.

GXGetFormatPaperType

You can use the `GXGetFormatPaperType` function to obtain the paper-type object referenced by a format object.

```
gxPaperType GXGetFormatPaperType (gxFormat aFormat);
```

`aFormat` A reference to the format object for which to obtain the paper-type object.

function result A reference to a paper-type object.

DESCRIPTION

The `GXGetFormatPaperType` function returns a reference to a paper-type object as its function result.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

SEE ALSO

Listing 3-16 on page 3-59 shows an example that uses the `GXGetFormatPaperType` function.

GXGetFormatForm

You can use the `GXGetFormatForm` function to retrieve the form and mask shapes for a particular format object.

```
gxShape GXGetFormatForm (gxFormat aFormat, gxShape *mask);
```

`aFormat` A reference to the format object associated with the form and mask shapes.

`mask` On return, the mask assigned to a format object.

function result A shape that represents the form.

DESCRIPTION

To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function. To replace any form previously associated with a particular format object, you use the `GXSetFormatForm` function. Picture shapes used by the form are flattened to disk with the format object during spooling.

RESULT CODES

`gxSegmentLoadFailedErr` A required code segment could not be found, or there was not enough memory to load it.

SEE ALSO

To associate a form with a format object, see the description of the `GXSetFormatForm` function in the next section.

GXSetFormatForm

You can use the `GXSetFormatForm` function to associate the form and mask shapes with a specific format object.

```
void GXSetFormatForm (gxFormat aFormat, gxShape form,
                      gxShape mask);
```

`aFormat` A reference to the format object in which to associate the form and mask shapes.

`form` A reference to a picture shape that specifies the form to assign to a format object.

Page Formatting and Dialog Box Customization

mask A reference to a picture shape that specifies the mask to assign to a format object.

DESCRIPTION

The `GXSetFormatForm` function replaces any form previously associated with a particular format object. It increments the owner counts of the new picture shapes (by calling the `GXCloneShape` function) and decrements the owner counts of the old picture shapes (by calling the `GXDisposeShape` function).

You may set either the `form` parameter or the `mask` parameter to `nil`.

Picture shapes are flattened to disk with the format object during spooling. To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function.

RESULT CODES

`gxSegmentLoadFailedErr` A required code segment could not be found, or there was not enough memory to load it.

SEE ALSO

Listing 3-12 on page 3-51 shows how to use the `GXSetFormatForm` function to associate the form and mask shapes with a specific format object.

To obtain the form shape associated with a format object, see the description of the `GXGetFormatForm` function in the previous section.

GXChangedFormat

You can use the `GXChangedFormat` function each time you change a format without directly calling `QuickDraw GX`.

```
void GXChangedFormat (gxFormat aFormat);
```

aFormat A reference to the format object which you are changing.

DESCRIPTION

You need to call the `GXChangedFormat` function each time you change a format object indirectly. For example, you should call this function when you modify a format collection.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

Displaying the Custom Page Setup Dialog Box

To allow a user to change a format object’s settings, you need to display the Custom Page Setup dialog box. You use the `GXFormatDialog` function to display the Custom Page Setup dialog box on the user’s screen.

GXFormatDialog

You can use the `GXFormatDialog` function to display the Custom Page Setup dialog box when the user chooses the Custom Page Setup menu item from the File menu.

```
gxDialogResult GXFormatDialog (gxFormat aFormat,
                               gxEditMenuRecord *anEditMenuRecord,
                               StringPtr title);
```

- `aFormat` A reference to the format object that specifies the values to display in the dialog box.
- `title` The title of the dialog box.
- `anEditMenuRecord` A structure for your application’s Edit menu and its menu items.

function result The user’s response to the dialog box.

DESCRIPTION

After you use the `GXFormatDialog` function to display the Custom Page Setup dialog box, the user can specify formatting information for a format (which is not the default format). For example, the user can specify the paper size, orientation, and the default formatting printer.

In the `anEditMenuRecord` parameter you to specify an Edit menu structure to support the standard editing operations of cut, copy, paste, and clear in dialog boxes.

The `GXFormatDialog` function returns a response that is defined in a dialog box result enumeration. If the user chooses the Format button, the `GXFormatDialog` function returns `gxOKSelected`. If the user chooses the Cancel button, the function returns `gxCancelSelected`. If the user chooses the Remove button, the function returns `gxRevertSelected`.

If an error occurs, the function returns `gxCancelSelected`. Call the `GXGetJobError` function to determine which error occurred.

Page Formatting and Dialog Box Customization

This function causes QuickDraw GX to send the `gxFormatDialog` message, which you can override to customize the Custom Page Setup dialog box.

Note that QuickDraw GX stores a user's responses to some dialog box items in the Custom Page Setup dialog box in a format collection.

SEE ALSO

Listing 3-9 on page 3-42 shows how to use the `GXFormatDialog` function to display the Custom Page Setup dialog box.

The Edit menu structure and the dialog box result enumeration are described in the chapter "Core Printing Features" in this book.

For information about customizing the Custom Page Setup dialog box, see "Adding Panels to Dialog Boxes" beginning on page 3-67.

Working With Panels

The following functions allow you to add panels to a dialog box. The `GXSetupDialogPanel` function adds a panel to a dialog box.

You use the `GXGetJobPanelDimensions` function to obtain the dimensions of a panel. This function allows you to locate the position of the cursor within a dialog panel.

You use the `GXEnableJobScalingPanel` function to prevent the display of the default scaling field in the Page Setup and Custom Page Setup dialog boxes. For example, if you implement your own scaling panel, you would disable the default scaling field provided by QuickDraw GX.

You typically call these methods from within an override function for the message that displays the panel. See the section "Message Override Functions for Customizing QuickDraw GX Dialog Boxes" beginning on page 3-119 for information about these message override functions.

GXSetupDialogPanel

You can use the `GXSetupDialogPanel` function to add a panel to a print dialog box.

```
OSErr GXSetupDialogPanel (gxPanelSetupRecord *panelRec);
```

`panelRec` A pointer to a panel setup structure.

function result An error code. The value `noErr` indicates that the operation was successful.

DESCRIPTION

The `GXSetupDialogPanel` function adds a panel, as defined by the information in the panel setup structure, to a print dialog box. You call this function from within your override of the `gxJobPrintDialog`, `gxFormatDialog`, and `gxJobDefaultFormatDialog` messages, before forwarding the message.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.
<code>gxCantAddPanelsNowErr</code>	Panels can only be added to a dialog box when the current driver is switched. This error is generated if a panel addition request is made at any other time.
<code>gxBadxdtlKeyErr</code>	An unknown key value was specified for an item in an extended dialog control resource.
<code>gxXdtlItemOutOfRangeErr</code>	An item referenced by the panel does not belong to the panel.
<code>gxNoActionButtonErr</code>	The action button for the panel is <code>nil</code> .
<code>gxTitlesTooLongErr</code>	The length of the button titles exceeds the maximum width allowed for a printing alert.

SEE ALSO

Listing 3-22 on page 3-68 shows how to use the `GXSetupDialogPanel` function to add a panel to the Custom Page Setup dialog box.

GXGetJobPanelDimensions

You can use the `GXGetJobPanelDimensions` function to obtain the dimensions of the area for the panel within a dialog box.

```
void GXGetJobPanelDimensions (gxJob aJob, Rect *aRect);
```

<code>aJob</code>	A reference to the job object associated with the panel.
<code>aRect</code>	On return, the rectangle whose geometry specifies the panel's size.

DESCRIPTION

When you want to locate the position of the cursor within a panel, you use the `GXGetJobPanelDimensions` function to obtain the dimensions of a panel.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

GXEnableJobScalingPanel

You can use the `GXEnableJobScalingPanel` function to prevent display of the default QuickDraw GX scaling field in the Page Setup and Custom Page Setup dialog boxes.

```
void GXEnableJobScalingPanel (gxJob aJob, Boolean enabled);
```

<code>aJob</code>	A reference to the job object associated with the scaling field.
<code>enabled</code>	A Boolean value that specifies whether or not to enable the scaling field.

DESCRIPTION

The `GXEnableJobScalingPanel` function enables or disables the scaling field. You set the `enabled` parameter to `true` to enable the scaling field and `false` to disable it. For example, you might disable this field if you want to provide your own scaling panel instead of the default field. The scaling field is enabled by default.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

GXGetMessageHandlerResFile

You can use the `GXGetMessageHandlerResFile` function to retrieve the resource file reference number of the printing extension or printer driver.

```
short GXGetMessageHandlerResFile (void);
```

function result The resource file reference number of the printing extension or printer driver.

DESCRIPTION

The `GXGetMessageHandlerResFile` function returns the resource file reference number for the printing extension or printer driver. You can use this function if you need to access data from these resources.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

Accessing Printing-Related Collection Objects

When you want to obtain a collection object associated with a particular job object, format object, or paper-type object, you use the `GXGetJobCollection`, `GXGetFormatCollection`, and `GXGetPaperTypeCollection` functions. For more information about collections, see the Collection Manager chapter of *Inside Macintosh: Environment and Utilities*.

GXGetJobCollection

You can use the `GXGetJobCollection` function to obtain the job collection object associated with a particular job object.

```
Collection GXGetJobCollection (gxJob aJob);
```

`aJob` A reference to the job object whose collection object you want to obtain.

function result A reference to a collection object.

DESCRIPTION

After you call the `GXGetJobCollection` function to obtain a job collection object, you must call the `GXGetJobError` function to obtain errors. It is important that you resolve errors immediately because the Collection Manager cannot work with a `nil` collection object.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

SEE ALSO

Listing 3-3 on page 3-28 shows how to use the `GXGetJobCollection` function to obtain the collection object associated with a particular job object.

GXGetFormatCollection

You can use the `GXGetFormatCollection` function to obtain the format collection object associated with a particular format object.

```
Collection GXGetFormatCollection (gxFormat aFormat);
```

`aFormat` A reference to the format object whose collection object you want to obtain.

function result A reference to a collection object.

DESCRIPTION

After you call the `GXGetFormatCollection` function to obtain a format collection object, you must call the `GXGetJobError` function to obtain errors. It is important that you resolve errors immediately because the Collection Manager cannot work with a `nil` collection object.

RESULT CODES

`gxSegmentLoadFailedErr` A required code segment could not be found, or there was not enough memory to load it.

GXGetPaperTypeCollection

You can use the `GXGetPaperTypeCollection` function to obtain the paper-type collection object associated with a particular paper-type object.

```
Collection GXGetPaperTypeCollection (gxPaperType aPaperType);
```

`aPaperType` A reference to the paper-type object whose collection object you want to obtain.

function result A reference to a collection object.

DESCRIPTION

After you call the `GXGetPaperTypeCollection` function to obtain a paper-type collection object, you must call the `GXGetJobError` function to obtain errors. It is important that you resolve errors immediately because the Collection Manager cannot work with a `nil` collection object.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

Application-Defined Functions

The following sections describe the functions that you must provide if you want to override QuickDraw GX print dialog messages or manipulate the format objects associated with a job object.

Message Override Functions for Customizing QuickDraw GX Dialog Boxes

To install an override function for a message, you need to call the `GXInstallApplicationOverride` function. Within the `GXInstallApplicationOverride` function, you specify a pointer to a function that you use to override a particular message for a specific dialog box. These messages include

- `gxJobPrintDialog`
- `gxJobDefaultFormatDialog`
- `gxFormatDialog`
- `gxHandlePanelEvent`
- `gxFilterPanelEvent`
- `gxParsePageRange`

You can use dialog box messages to invoke your actions. QuickDraw GX sends the `gxJobDefaultFormatDialog` message when your application calls `GXJobDefaultFormatDialog` to display the Page Setup dialog box. QuickDraw GX sends the `gxFormatDialog` message when your application calls `GXFormatDialog` to display the Custom Page Setup dialog box and it sends the `gxJobPrintDialog` message when your application calls `GXJobPrintDialog` to display the Print dialog box.

QuickDraw GX also sends the `gxHandlePanelEvent` message and the `gxFilterPanelEvent` message when an event occurs in a panel.

GXJobPrintDialog

QuickDraw GX sends the `gxJobPrintDialog` message when the application calls `gxJobPrintDialog` to display the Print dialog box. You can install an override function for the `gxJobPrintDialog` message to modify the behavior or appearance of the Print dialog box. Your override function must match the following formal declaration:

```
OSErr GXJobPrintDialog (gxDialogResult *aDialogResult);
```

`aDialogResult`

On return, a pointer to the selection made by the user in the dialog box.

function result An error code. The value `noErr` indicates that the operation was successful.

DESCRIPTION

QuickDraw GX sends the `gxJobPrintDialog` message when the user selects Print from the File menu and the application subsequently calls the `GXJobPrintDialog` function to display the Print dialog box on the user's screen.

The default implementation of this message adds the standard printing panels and interface and then displays the dialog box.

You usually override this message to customize the dialog box by adding panels using the `GXSetupDialogPanel` function.

SPECIAL CONSIDERATIONS

You never send the `gxJobPrintDialog` message yourself.

You must forward the `gxJobPrintDialog` message to other message handlers. Add your panels and then forward the message.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The user has canceled printing.

GXJobDefaultFormatDialog

QuickDraw GX sends the `gxJobDefaultFormatDialog` message when the application calls the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box. You can install an override function for the `gxJobDefaultFormatDialog` message to modify the behavior or appearance of the dialog box. Your override function must match the following formal declaration:

```
OSErr GXJobDefaultFormatDialog (gxDialogResult *aDialogResult);
```

`aDialogResult`

On return, a pointer to a value that specifies the selection made by the user in the dialog box.

function result An error code. The value `noErr` indicates that the operation was successful.

DESCRIPTION

QuickDraw GX sends the `gxJobDefaultFormatDialog` message when the user clicks the More Choices button in the Page Setup dialog box. The application calls the `GXJobDefaultFormatDialog` function to display the extended Page Setup dialog box.

The default implementation of this message adds the standard printing panels and interface and then displays the dialog box.

You usually override this message to customize the dialog box by adding panels using the `GXSetupDialogPanel` function. You can add your own panels to the dialog box through the normal QuickDraw GX printing calls.

SPECIAL CONSIDERATIONS

You never send the `gxJobDefaultFormatDialog` message yourself.

You must forward the `gxJobDefaultFormatDialog` message to other message handlers. Add your panels and then forward the message.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

GXFormatDialog

QuickDraw GX sends the `gxFormatDialog` message when the application calls the `GXFormatDialog` function to display the Custom Page Setup dialog box. You can install an override function for the `gxFormatDialog` message to modify the behavior or appearance of the dialog box. Your override function must match the following formal declaration:

```
OSErr GXFormatDialog (gxFormat aFormat, StringPtr title,
                     gxDialogResult *aDialogResult);
```

`aFormat` A reference to the format object.

`title` The title of the dialog box. If you specify `nil` as the value of this parameter, the title “Custom Page Setup” is used.

`aDialogResult` On return, a pointer to the selection made by the user in the dialog box.

function result An error code. The value `noErr` indicates that the operation was successful.

DESCRIPTION

QuickDraw GX sends the `gxFormatDialog` message when the user selects the Custom Page Setup menu item and an application subsequently calls the `GXFormatDialog` function to display the Custom Page Setup dialog box.

The default implementation of this message adds the standard printing panels and interface and then displays the dialog box.

You usually override this message to customize the dialog box by adding panels using the `GXSetupDialogPanel` function.

SPECIAL CONSIDERATIONS

You never send the `gxFormatDialog` message yourself.

You must forward the `gxFormatDialog` message to other message handlers. Add your panels and then forward the message.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

GXHandlePanelEvent

QuickDraw GX sends the `gxHandlePanelEvent` message when an event happens in a panel. You can install an override function for the `gxHandlePanelEvent` message to handle panel events that cannot be handled using extended item list resources. Your override function must match the following formal declaration:

```
OSErr GXHandlePanelEvent (gxPanelInfoRecord *aPanelInfoRecord,
                          gxPanelResult *panelResult);
```

`aPanelInfoRecord`

A pointer to the panel information structure that supplies information to the panel about the current dialog box and panel event.

`panelResult`

On return, the result of handling the panel event.

function result An error code. The value `noErr` indicates that the operation was successful.

DESCRIPTION

QuickDraw GX sends the `gxHandlePanelEvent` message to allow a panel to handle events associated with the dialog box. The result code returned by the `panelResult` parameter is either a value of type `OSErr`, or one of the following values:

`gxPanelNoResult`

The returned value does not currently have any meaning.

`gxPanelCancelConfirmation`

The user confirmed the panel, however, the panel handler discovers that the user entered an inappropriate value in the dialog box.

The default implementation of this message does nothing. You need to override this message if you add panels that cannot be handled using extended item list resources.

SPECIAL CONSIDERATIONS

You never send the `gxHandlePanelEvent` message yourself.

You always perform a total override of the `gxHandlePanelEvent` message, in which you handle any events of interest that occur in your panel.

RESULT CODES

`gxSegmentLoadFailedErr`

A required code segment could not be found, or there was not enough memory to load it.

`gxPrUserAbortErr`

The user has canceled printing.

GXFilterPanelEvent

QuickDraw GX sends the `gxFilterPanelEvent` message when an event happens in a panel. You can install an override function for the `gxFilterPanelEvent` message to add panels that need a filter procedure. Your override function must match the following formal declaration:

```
OSErr GXFilterPanelEvent (gxPanelInfoRecord *aPanelInfoRecord;
                        Boolean *returnImmed);
```

`aPanelInfoRecord`

A pointer to the panel information structure that supplies information to the panel about the current dialog box and panel event.

`returnImmed`

On return, a Boolean value that is `true` if there should be no further processing on this event and `false` if not.

function result An error code. The value `noErr` indicates that the operation was successful.

DESCRIPTION

QuickDraw GX sends the `gxFilterPanelEvent` message to filter panel events in a dialog box.

The default implementation of this message does nothing. You need to override this message if you add panels that require a filtering process.

SPECIAL CONSIDERATIONS

You never send the `gxFilterPanelEvent` message yourself.

You always perform a total override of the `gxFilterPanelEvent` message, in which you filter any events that occur in your panel.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

GXParsePageRange

QuickDraw GX sends the `gxParsePageRange` message when the user selects a range of pages for printing. You can install an override function for the `gxParsePageRange` message to modify or validate the page range. Your override function must match the following formal declaration:

```
OSErr GXParsePageRange (StringPtr fromString, StringPtr toString,
                        gxParsePageRangeResult *result);
```

`fromString` A pointer to the string representation of the From page.

`toString` A pointer to the string representation of the To page.

`result` On return, a value that specifies the result code for the range parsing. The constants for this value are given in the section “The Panel Setup Structure” on page 3-101.

function result An error code. The value `noErr` indicates that the operation was successful.

DESCRIPTION

QuickDraw GX sends the `gxParsePageRange` message to validate that a page range entered by the user is appropriate for the print job.

The default implementation of this message adds the standard printing panels and interface and then displays the dialog box.

You usually override this message to customize the dialog box by adding panels using the `GXSetupDialogPanel` function.

SPECIAL CONSIDERATIONS

You rarely need to send the `gxParsePageRange` message yourself.

You must forward the `gxParsePageRange` message to other message handlers.

RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

Looping Through Format Objects

When you want to make changes to each format object associated with a document, you can use the `GXForEachJobFormatDo` function to access format objects. In this function you must provide a pointer to a format function.

To access each format object associated with a printable document, provide a pointer to a format function in the `GXForEachJobFormatDo` function that takes two parameters: the format object associated with a particular job object, and a pointer to a reference constant in which you specify unique format object references. For example, this is how you should declare the function if you were to name it `MyFormatFunction`:

```
gxLoopStatus MyFormatFunction (gxFormat aFormat, void *refCon);
```

`aFormat` The current format. This is provided by QuickDraw GX when the function is called.

`refCon` A pointer to a reference constant for each format object.

function result A Boolean value to indicate whether looping should stop.

DESCRIPTION

When you use the `GXForEachJobFormatDo` function, QuickDraw GX calls your format function multiple times as it retrieves each format object referenced by a job object.

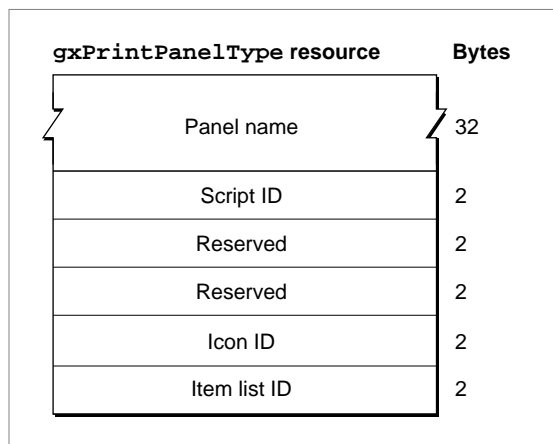
Dialog Box-Related Resources

This section describes the resources that you use when you add panels to QuickDraw GX dialog boxes.

The Panel Resource

Figure 3-20 shows the format of the compiled panel resource, `gxPrintPanelType`.

Figure 3-20 Panel resource



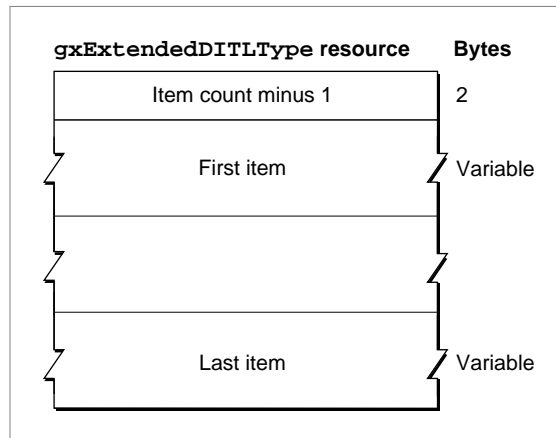
The compiled version of a panel resource contains the following elements:

- Panel name. This is a Pascal string that contains the name of the panel.
- Script ID. This is the name of the script in which the panel is written; for example, `smRoman`.
- Reserved. These words are reserved for future use.
- Icon ID. This is the resource ID for the icon resource that displays in the expanded dialog box.
- Item ID. This is the resource ID of the items that are displayed in the panel.

The Extended Item List Resource

Figure 3-21 shows the format of the compiled extended item resource, `gxExtendedDITLType`.

Figure 3-21 Extended item list resource



The compiled version of this resource contains the following elements:

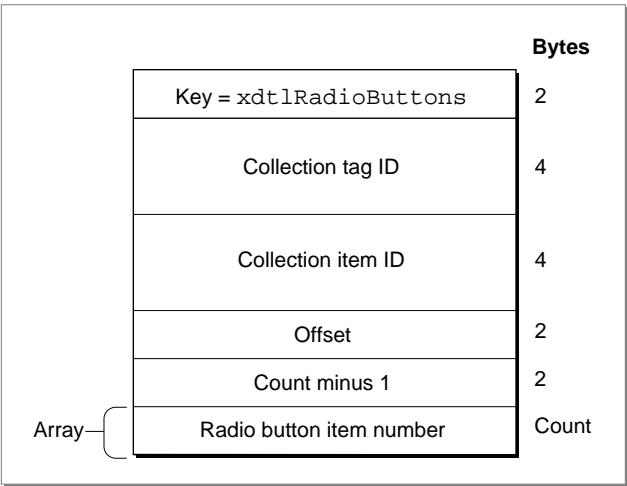
- Item count - 1. This is the number of items in the resource, less 1.
- A variable number of items.

The format of each item depends on its type, as defined below.

<code>xdtlRadioButtons</code>	Radio buttons
<code>xdtlCheckBox</code>	Checkbox
<code>xdtlEditTextInteger</code>	Integer-format editable text
<code>xdtlEditTextReal</code>	Real-format editable text
<code>xdtlEditTextString</code>	String-format editable text
<code>xdtlPopup</code>	Pop-up menu

The compiled version of an item for a group of radio buttons is shown in Figure 3-22.

Figure 3-22 Radio button items

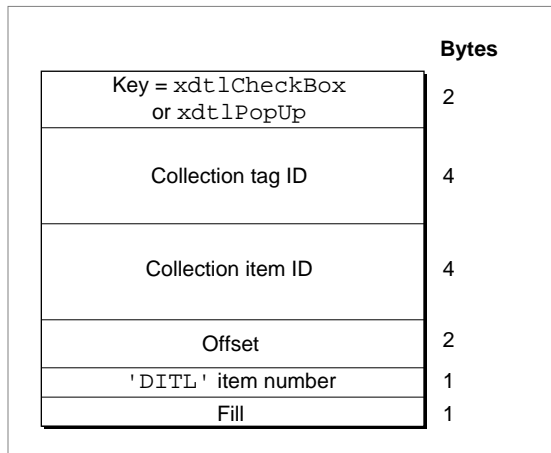


It contains the following elements:

- **Key.** The key specifies the kind of item. It is always `xdtlRadioButtons`.
- **Collection tag ID.** The collection tag specifies the creator of the collection item, such as `gxPrintingTagID` for items provided by QuickDraw GX in the job, format, and paper-type collections.
- **Collection item ID.** The item ID specifies the collection item ID, such as `'incf'` for the level of fonts to include.
- **Offset.** The offset specifies the start of storage for the data. It is the number of bytes into the collection item.
- **Count.** The count specifies the number of radio buttons in this list. Because there is 1 byte per button in the collection item, the count also specifies the size for the group of buttons in the collection item.
- **Item number.** The item number specifies the item list's item that corresponds to this radio button. There is one item per button.

The compiled version of an item for both a checkbox or pop-up menu is shown in Figure 3-23.

Figure 3-23 Checkbox and pop-up menu items

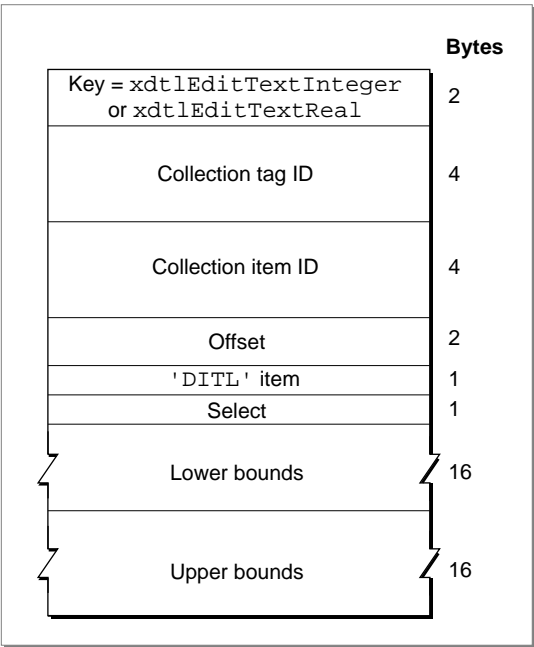


It contains the following elements:

- **Key.** The key specifies the kind of item. It is always `xdt1CheckBox` for checkboxes and `xdt1PopUp` for pop-up menus.
- **Collection tag ID.** The collection tag specifies the creator of the collection item, such as `gxPrintingTagID` for items provided by QuickDraw GX in the job, format, and paper-type collections.
- **Collection item ID.** The item ID specifies the collection item ID, such as `'dest'` for whether to print to a file.
- **Offset.** The offset specifies the start of storage for the data. It is the number of bytes into the collection item.
- **Item number.** The item number specifies the item list's item that corresponds to this checkbox or pop-up menu.
- **Fill.** The fill is 1 byte.

The compiled version of an item for both integer or real editable text is shown in Figure 3-24.

Figure 3-24 Integer and real edit text items



It contains the following elements:

- **Key.** The key specifies the kind of item. It is always `xdtlEditTextInteger` for integers and `xdtlEditTextReal` for real numbers.
- **Collection tag ID.** The collection tag specifies the creator of the collection item, such as `gxPrintingTagID` for items provided by QuickDraw GX in the job, format, and paper-type collections.
- **Collection item ID.** The item ID specifies the collection item ID, such as `'copy'` for the number of copies.
- **Offset.** The offset specifies the start of storage for the data. It is the number of bytes into the collection item.
- **Item number.** The item number specifies the item list's item that corresponds to the editable text item.
- **Select.** This element specifies whether or not to highlight the text. If its value is 0, the text is not highlighted. If its value is 1, the text is highlighted.
- **Lower bounds.** This is a Pascal string that contains an optional sign (plus or minus), digits, and for real numbers, an optional decimal point before the fractional part of the number. If the string is `nil`, no lower bounds is specified.
- **Upper bounds.** This is a Pascal string that contains an optional sign (plus or minus), digits, and for real numbers, an optional decimal point before the fractional part of the number. If the string is `nil`, no upper bounds is specified.

The compiled version of an item for string editable text is shown in Figure 3-25.

Figure 3-25 String editable text items

	Bytes
Key = <code>xdtlEditTextString</code>	2
Collection tag ID	4
Collection item ID	4
Offset	2
'DITL' item	1
Select	1

It contains the following elements:

- **Key.** The key specifies the kind of item. It is always `xdtlEditTextString`.
- **Collection tag ID.** The collection tag specifies the creator of the collection item, such as `gxPrintingTagID` for items provided by QuickDraw GX in the job, format, and paper-type collections.
- **Item ID.** The item ID specifies the collection item ID, such as `'incf'` for the level of fonts to include.
- **Offset.** The offset specifies the start of storage for the data. It is the number of bytes into the collection item.
- **Item number.** The item number specifies the item list's item that corresponds to the editable text item.
- **Select.** This element specifies whether or not to highlight the text. If its value is 0, the text is not highlighted. If its value is 1, the text is highlighted.

Summary of Page Formatting and Dialog Box Customization

Constants and Data Types

Constants for Loop Status Information

```
enum {
    gxStopLooping = false, /* stop looping */
    gxKeepLooping = true   /* keep looping */
};

typedef Boolean gxLoopStatus;

/* function for each format object associated with a job object */
typedef gxLoopStatus (*gxFormatProc) (gxFormat aFormat, void *refCon);
```

Constants for Collection Item Categories and Tag IDs

Collection Item Categories

```
typedef short gxCollectionCategory; /* stored in collection object items' */
/* user attribute bits */

enum {
    gxNoCollectionCategory= (gxCollectionCategory) 0x0000, /* not volatile */
    gxOutputDriverCategory= (gxCollectionCategory) 0x0001, /* affected by out-
                                                             put printer */
    gxFormattingDriverCategory= (gxCollectionCategory) 0x0002, /* affected by
                                                             formatting
                                                             printer */
    gxDriverVolatileCategory= (gxCollectionCategory) 0x0004, /* volatile */

    gxVolatileOutputDriverCategory =
        /* purge when output printer driver changes */
        gxOutputDriverCategory + gxDriverVolatileCategory,
```

```

gxVolatileFormattingDriverCategory =
    /* purge when formatting printer driver changes */
    gxFormattingDriverCategory + gxDriverVolatileCategory
};

```

Collection Tag ID

```

enum { gxPrintingTagID = -28672 }; /* QuickDraw GX assigns its collection
                                   objects with the same 4-byte ID */

```

Constants and Data Types for Job Collection Items

Print-Job Information

```

enum { gxJobTag = 'job ' }; /* item ID for the print-job item */

/* job object information structure */
struct gxJobInfo {
    long    numPages;          /* total number of pages to print */
    long    priority;          /* print job's priority */
    long    timeToPrint;       /* designated time to print a print job */
    long    jobTimeout;        /* time to cancel print job, in ticks */
    long    firstPageToPrint   /* first page to begin printing */
    short   jobAlert;          /* when to alert the user about printing */
    Str31   appName;           /* name of application used to create the */
                                   /* printable document */
    Str31   documentName;      /* name of the user's document */
    Str31   userName;          /* name of the user associated with the */
                                   /* printable document */
};

enum {
    /* print-job priorities */
    gxPrintJobUrgent = 0x00000001, /* priority of print job is */
                                   /* "urgent" */
    gxPrintJobAtTime = 0x00000002, /* designated time to print the */
                                   /* print job */
    gxPrintJobASAP = 0x00000003 /* designated time to print the */
                                   /* print job is "as soon as */
                                   /* possible" */
};

```

```

enum { gxPrintJobHoldingBit = 0x00001000 }; /* reserved bit in the
                                             priority field indicates a
                                             print job on hold */

enum {
    /* print-job holding status */
    gxPrintJobHolding      = (gxPrintJobHoldingBit + gxPrintJobASAP),
    gxPrintJobHoldingAtTime = (gxPrintJobHoldingBit + gxPrintJobAtTime),
    gxPrintJobHoldingUrgent = (gxPrintJobHoldingBit + gxPrintJobUrgent)
};

enum {
    /* print-job alert constants */
    gxNoPrintTimeAlert= 0,    /* don't alert user when printing */
    gxAlertBefore      = 1,    /* alert user before printing */
    gxAlertAfter       = 2,    /* alert user after printing */
    gxAlertBothTimes   = 3     /* alert user before and after printing */
};

enum {
    /* time to cancel print job */
    gxThirtySeconds    = 1800, /* cancel print job in 30 seconds (in ticks) */
    gxTwoMinutes       = 7200  /* cancel print job in 2 minutes (in ticks) */
};

```

Collation Information

```

enum { gxCollationTag = 'sort' }; /* item ID for the collation item*/

/* collation information structure */
struct gxCollationInfo {
    Boolean collation;          /* indicates whether or not to collate */
                                /* copies */
};

```

Copies Information

```

enum { gxCopiesTag = 'copy' }; /* item ID for the copies item*/

/* copies information structure */
struct gxCopiesInfo {
    long copies;                /* number of copies of a document to print */
};

```

Page-Range Information

```

enum { gxPageRangeTag = 'rang' }; /* item ID for the page-range item */

/* page-range information structure */
struct gxPageRangeInfo {
    gxSimplePageRangeInfo    simpleRange;    /* simple page range */
                                           /* information */
    Str31                    fromString;      /* beginning of customized */
                                           /* page range */
    Str31                    toString;        /* end of customized page */
                                           /* range */
    long                     minFromPage;     /* minimum of default page */
                                           /* range */
    long                     maxToPage;       /* maximum of default page */
                                           /* range */
    char                     replaceString[1]; /* page-range replacement */
                                           /* string */
};

/* simple page-range information structure */
struct gxSimplePageRangeInfo {
    char    optionChosen; /* specific page-range option */
    Boolean printAll;     /* true if user wants to print all pages of a */
                        /* document */
    long    fromPage;     /* first page in page range */
    long    toPage;       /* last page in page range */
};

enum {
    /* page-range options */
    gxDefaultPageRange = (char) 0, /* use default numeric page range */
    gxReplacePageRange = (char) 1, /* use editable text field */
    gxCustomizePageRange = (char) 2 /* use alphanumeric page range */
};

```

Quality Information

```
enum { gxQualityTag = 'qual' }; /* item ID for the quality item*/

/* quality information structure */
struct gxQualityInfo {
    Boolean    disableQuality; /* true to disable standard quality */
                                /* controls */
    short      defaultQuality; /* default quality */
    short      currentQuality; /* current quality */
    short      qualityCount;   /* number of quality menu items in */
                                /* Quality pop-up menu */
    char       qualityNames[1]; /* Quality pop-up menu names, such as */
                                /* "Best" */
};
```

File-Destination Information

```
enum { gxFileDestinationTag = 'dest' }; /* item ID for the file- */
                                         /* destination item*/

/* file-destination information structure */
struct gxFileDestinationInfo {
    Boolean toFile; /* true if destination is a file */
};
```

File-Location Information

```
enum { gxFileLocationTag = 'floc' } /* item ID for the file- */
                                     /* location item*/

/* file-location information structure */
struct gxFileLocationInfo {
    FSSpec fileSpec; /* location of file */
};
```

File-Format Information

```
enum { gxFileFormatTag = 'ffmt' }; /* item ID for the file- */
                                     /* format item*/
```

```

/* file-format information structure */
struct gxFileFormatInfo {
    Str31 fileFormatName;          /* name of file format */
};

```

File-Fonts Information

```

enum { gxFileFontsTag = 'incf' }; /* item ID for the file-fonts item */

/* file-fonts information structure */
struct gxFileFontsInfo {
    char          includeFonts;    /* font include level; if destination is
                                   file */
};

enum {                                /* font include levels */

    gxIncludeNoFonts      = (char) 1,
    gxIncludeAllFonts     = (char) 2,
    gxIncludeNonStandardFonts = (char) 3
};

```

Paper-Feed Information

```

enum { gxPaperFeedTag = 'feed' }; /* item ID for paper-feed item */

/* paper-feed information structure */
struct gxPaperFeedInfo {
    Boolean      autoFeed;        /* true if automatic feed, false if */
                                   /* manual feed */
};

```

Manual-Feed Information

```

enum { gxManualFeedTag = 'manf' }; /* item ID for manual-feed item */

/* manual-feed information structure */
struct gxManualFeedInfo {
    long         numPaperTypeNames; /* number of paper-type objects to
                                   /* manually feed */
    Str31        paperTypeNames[1]; /* names of paper-type objects to */
                                   /* manually feed */
};

```

Standard Mapping Information

```
enum { gxNormalMappingTag = 'nmap' }; /* item ID for the standard */
                                     /* mapping item */

/* standard mapping information structure */
struct gxNormalMappingInfo {
    Boolean      normalPaperMapping; /* true if not overriding standard */
                                     /* paper matching */
};
```

Special Mapping Information

```
enum { gxSpecialMappingTag = 'smap' }; /* item ID for special mapping */

/* special mapping information structure */
struct gxSpecialMappingInfo {
    char          specialMapping;      /* specific paper-mapping option */
} ;

enum {
    /* paper-mapping options */
    gxRedirectPages = (char) 1,      /* user wants to crop redirected pages */
    gxScalePages    = (char) 2,      /* user wants to scale pages */
    gxTilePages     = (char) 3       /* user wants to tile pages */
};
```

Tray-Mapping Information

```
enum { gxTrayMappingTag = 'tmap' };

struct gxTrayMappingInfo{
    gxTrayIndex mapPaperToTray; /* tray to map all paper to */
};

typedef long gxTrayIndex;      /* specifies the paper tray setting */
```

Print-Panel Information

```
enum { gxPrintPanelTag = 'ppan' }; /* item ID for the Print */
                                     /* panel item */
```

Page Formatting and Dialog Box Customization

```

/* print-panel information structure */
struct gxPrintPanelInfo {
    Str31    startPanelName;          /* name of starting panel in
                                      /* Print dialog box */
};

```

Format-Panel Information

```

enum { gxFormatPanelTag = 'fpan' }; /* item ID for the format */
                                      /* panel item */

/* format-panel information structure */
struct gxFormatPanelInfo {          /* name of starting panel in */
    Str31    startPanelName;        /* Page Setup dialog box */
};

```

Paper-Mapping Information

```

enum { gxPaperMappingTag = 'pmap' }; /* item ID for print- */
                                      /* panel item */

/* This collection item contains a flattened paper-type object resource. */

```

Translated-Document Information

```

enum { gxTranslatedDocumentTag = 'trns' };

struct gxTranslatedDocumentInfo {
    long translatorInfo;             /* information from the translation process */
};

```

Constants and Data Types for Format Collection Items

Orientation Information

```

enum { gxOrientationTag = 'layo' }; /* item ID for the */
                                      /* orientation item */

/* orientation information structure */
struct gxOrientationInfo {
    char      orientation;           /* an enumerated orientation value */
};

```



```
enum {
    /* orientation options */
    gxPortraitLayout      = (char) 0, /* user wants portrait orientation */
    gxLandscapeLayout     = (char) 1, /* user wants landscape orientation */
    gxRotatedPortraitLayout = (char) 2, /* rotated portrait orientation,
                                         not user specifiable*/
    gxRotatedLandscapeLayout = (char) 3 /* user wants rotated landscape */
                                         /* orientation */
};
```

Scaling Information

```
enum { gxScalingTag = 'scal' }; /* item ID for the scaling item */

/* scaling information structure */
struct gxScalingInfo {
    Fixed    horizontalScaleFactor; /* current horizontal scaling */
                                         /* factor */
    Fixed    verticalScaleFactor; /* current vertical scaling factor */
    short    minScaling; /* minimum current scaling factor */
    short    maxScaling; /* maximum current scaling factor*/
};
```

Direct-Mode Information

```
enum { gxDirectModeTag = 'dirm' }; /* item ID for the direct- */
                                         /* mode item */

/* direct-mode information structure */
struct gxDirectModeInfo {
    Boolean    directModeOn; /* true if direct mode is enabled */
};
```

Format-Halftone Information

```
enum { gxFormatHalftoneTag = 'half' }; /* item ID for the special */
                                         /* mapping item */

/* format-halftone information structure */
struct gxFormatHalftoneInfo {
    long    numHalftones; /* number of halftones */
    gxHalftone    halftones[1]; /* any number of halftones */
};
```

Page-Inversion Information

```
enum { gxInvertPageTag = 'invp' }; /* item ID for the page- */
                                   /* inversion item */

/* page-inversion information structure */
struct gxInvertPageInfo {
    Boolean invert;                /* if true, invert the page */
};                                /* if missing or false, don't invert */
                                   /* the page */
```

Horizontal Page-Flip Information

```
enum { gxFlipPageHorizontalTag = 'flph' }; /* item ID for the */
                                             /* horizontal page-flip item */

/* horizontal flip-page information structure */
struct gxFlipPageHorizontalInfo{
    Boolean flipHorizontal;        /* if true, flip x coordinates on the */
};                                /* page; if missing or false, don't flip */
```

Vertical Page-Flip Information

```
enum { gxFlipPageVerticalTag = 'flpv' }; /* item ID for the */
                                             /* vertical page-flip item */

/* vertical page-flip information structure */
struct gxFlipPageVerticalInfo {
    Boolean flipVertical;          /* if true, flip y coordinates on the */
};                                /* page; if missing or false, don't flip */
```

Precise-Bitmap Information

```
enum { gxPreciseBitmapsTag = 'pbmp' }; /* item ID for the precise- */
                                             /* bitmap item */

/* precise-bitmap information structure */
struct gxPreciseBitmapInfo {
    Boolean preciseBitmaps;        /* if true, scale the page by 96% */
};                                /* if missing or false, don't scale */
```

Paper-Type Lock Information

```
enum { gxPaperTypeLockTag = 'ptlk' }; /* item ID for the paper- */
                                      /* type lock item*/

/* paper-type object lock information structure */
struct gxPaperTypeLockInfo {
    Boolean  paperTypeLocked;      /* true if paper-type object */
};                                /* is locked */
```

Constants and Data Types for Paper-Type Collection Items

Base Information

```
enum { gxBaseTag = 'base' };          /* item ID for the base item */

/* base type information structure */
struct gxBaseInfo {
    long  baseType;                  /* base type chosen */
} ;

enum {
    /* paper-type object base types */
    gxUnknownBase = 0,              /* unknown base type */
    gxUsLetterBase = 1,             /* US letter base type */
    gxUsLegalBase = 2,             /* US legal base type */
    gxA4LetterBase = 3,            /* A4 letter base type */
    gxB5LetterBase = 4,            /* B5 letter base type */
    gxTabloidBase = 5              /* tabloid base type */
};
```

Creator Information

```
enum { gxCreatorTag = 'crea' }; /* item ID for the creator item */

/* creator information structure */
struct gxCreatorInfo {
    OStype  creator;                /* creator of the paper-type object */
};

enum {
    /* paper-type object creator type */
    gxSysPaperType = 'sypt',        /* system paper-type object creator */
};
```

```

gxUserPaperType = 'uspt'      /* user paper-type object creator */
/* if printer driver creates a paper-type object, use printer
/* driver's creator type */
};

```

Units Information

```

enum { gxUnitsTag = 'unit' }; /* item ID for the units item */

/* unit information structure */
struct gxUnitsInfo {
    char    units;              /* specific paper-type object */
                                /* measurement */
};

enum {
    /* paper-type object units */
    gxPicas = (char) 0,        /* pica measurement */
    gxMms    = (char) 1,        /* millimeter measurement */
    gxInches = (char) 2        /* inches measurement */
};

```

Flags Information

```

enum { gxFlagsTag = 'flag' }; /* item ID for the flags item */

/* flags information structure */
typedef struct {
    long    flags;              /* paper-type object flags */
}gxFlagsInfo;

enum {
    /* paper-type object flags (bit positions) */
    gxOldPaperTypeFlag    = 0x00800000, /* indicates a paper-type object */
                                /* with 7.0 settings */
    gxNewPaperTypeFlag     = 0x00400000, /* indicates a paper-type object */
                                /* with post 7.0 settings */
    gxOldAndNewPaperTypeFlag = 0x00C00000, /* indicates a paper-type object */
                                /* that is both old and new */
    gxDefaultPaperTypeFlag = 0x00100000, /* indicates the default paper */
                                /* type */
};

```

Comment Information

```
enum { gxCommentTag = 'cmnt' }; /* item ID for the comment item */

/* comment information structure */
struct gxCommentInfo {
    Str255    comment;          /* paper-type object comment */
} ;
```

Panel-Related Constants and Data Types

QuickDraw GX Dialog Box Panel Information

```
/* constants for overriding messages when adding dialog box panels */
#define gxJobStatus                3
#define gxPrintingEvent            4
#define gxJobDefaultFormatDialog  5
#define gxFormatDialog             6
#define gxJobPrintDialog           7
#define gxFilterPanelEvent         8
#define gxHandlePanelEvent         9
#define gxParsePageRange          10

/* dialog box related resources */
#define gxXdtlRadioButtons         0
#define gxXdtlCheckBox             1
#define gxXdtlEditTextInteger     2
#define gxXdtlEditTextReal        3
#define gxXdtlEditTextString      4
#define gxXdtlPopUp               5
```

The Panel Information Structure

```
struct gxPanelInfoRecord {
    gxPanelEvent panelEvt; /* the event */
    short panelResId;      /* resource ID of current panel resource */
    DialogPtr pDlg;        /* pointer to dialog */
    EventRecord *theEvent; /* pointer to event */
    short itemHit;         /* actual item number of event */
    short itemCount;       /* number of items before your items */
    short evtAction;        /* the action that will occur after
                           this event is processed */
    short errorStringId;    /* 'STR ' ID of error string */
}
```

Page Formatting and Dialog Box Customization

```

    gxFormat theFormat;      /* the current format */
    void *refCon;            /* refCon from gxPanelSetupRecord */
};

```

Panel Events

```

enum {
    gxPanelNoEvt      = (gxPanelEvent) 0,      /* no event */
    gxPanelOpenEvt    = (gxPanelEvent) 1,      /* panel is about to open */
    gxPanelCloseEvt   = (gxPanelEvent) 2,      /* panel is about to close */
    gxPanelHitEvt     = (gxPanelEvent) 3,      /* user has selected item */
    gxPanelActivateEvt= (gxPanelEvent) 4,      /* panel has been activated */
    gxPanelDeactivateEvt= (gxPanelEvent) 5,    /* panel has been deactivated */
    gxPanelIconFocusEvt= (gxPanelEvent) 6,     /* focus has changed to icons */
    gxPanelPanelFocusEvt= (gxPanelEvent) 7,    /* focus has changed to panel */
    gxPanelFilterEvt  = (gxPanelEvent) 8,      /* panel event needs to be
                                                filtered */
    gxPanelCancelEvt  = (gxPanelEvent) 9,      /* panel has been canceled */
    gxPanelConfirmEvt = (gxPanelEvent) 10,     /* panel has been confirmed */
    gxPanelDialogEvt  = (gxPanelEvent) 11,     /* panel event to be handled
                                                by the dialog box handler */
    gxPanelOtherEvt   = (gxPanelEvent) 12,     /* an OS event has occurred
                                                in the panel */
    gxPanelUserWillConfirmEvt
                        = (gxPanelEvent) 13    /* user has selected OK */
};

```

```
typedef long gxPanelEvent;
```

Panel Responses

```

enum {
    gxPanelNoResult      = 0,  /* no result from panel */
    gxPanelCancelConfirmation = 1, /* user confirmed panel, but panel
                                    handler discovered an error */
};

```

```
typedef long gxPanelResult;
```

Panel Event Actions

```

enum {
    gxOtherAction      = 0,  /* current item doesn't change after event */
    gxClosePanelAction = 1,  /* panel is closed after event */
};

```

```

    gxCancelDialogAction = 2, /* dialog box is canceled after event */
    gxConfirmDialogAction= 3 /* dialog box is confirmed after event */
};

```

The Panel Setup Structure

```

struct gxPanelSetupRecord {
    gxPrintingPanelKind    panelKind;          /* kind of program using panel */
    short                  panelResId;         /* resource ID of panel */
    short                  resourceRefNum;     /* resource file refnum of panel */
    void                   *refCon;           /* pointer to panel setup
                                              structure used to build panel */
};

```

Printing Panel Kinds

```

enum {
    gxApplicationPanel= (gxPrintingPanelKind) 0, /* an application panel */
    gxExtensionPanel   = (gxPrintingPanelKind) 1, /* printing extension panel */
    gxDriverPanel      = (gxPrintingPanelKind) 2 /* printer driver panel */
};

typedef long gxPrintingPanelKind;

```

Parse Range Results

```

enum {
    gxRangeNotParsed   = (gxParsePageRangeResult) 0, /* not parsed yet */
    gxRangeParsed      = (gxParsePageRangeResult) 1, /* successful parse */
    gxRangeBadFromValue= (gxParsePageRangeResult) 2, /* the "from page" */
                                                         /* value is invalid */
    gxRangeBadToValue  = (gxParsePageRangeResult) 3 /* the "to page" */
                                                         /* value is invalid */
};

typedef long gxParsePageRangeResult;

```

Functions

Creating and Manipulating Format Objects

```

gxFormat GXNewFormat          (gxJob aJob);
void GXDisposeFormat          (gxFormat aFormat);
gxFormat GXCopyFormat         (gxFormat srcFormat, gxFormat dstFormat);

```

Page Formatting and Dialog Box Customization

```

gxFormat GXCloneFormat      (gxFormat aFormat);
long GXCountJobFormats      (gxJob aJob);
long GXCountFormatOwners    (gxFormat aFormat);
void GXForEachJobFormatDo    (gxJob aJob, gxFormatProc aFormatProc, void
                             *refCon);

```

Manipulating Format Object Properties

```

void GXGetFormatMapping      (gxFormat aFormat, gxMapping *aMapping);
gxPaperType GXGetFormatPaperType
                             (gxFormat aFormat);
gxShape GXGetFormatForm      (gxFormat aFormat, gxShape *mask);
void GXSetFormatForm         (gxFormat aFormat, gxShape form, gxShape mask);
void GXChangedFormat         (gxFormat aFormat);

```

Displaying the Custom Page Setup Dialog Box

```

gxDialogResult GXFormatDialog
                             (gxFormat aFormat,
                             gxEditMenuRecord *anEditMenuRecord,
                             StringPtr title);

```

Working With Panels

```

void GXSetupDialogPanel      (gxPanelSetupRecord *aPanelSetupRecord);
void GXGetJobPanelDimensions
                             (gxJob aJob, Rect *aRect);
void GXEnableJobScalingPanel
                             (gxJob aJob, Boolean enabled);
short GXGetMessageHandlerResFile
                             (void);

```

Accessing Printing-Related Collection Objects

```

Collection GXGetJobCollection
                             (gxJob aJob);
Collection GXGetFormatCollection
                             (gxFormat aFormat);
Collection GXGetPaperTypeCollection
                             (gxPaperType aPaperType);

```


Application-Defined Functions

Message Override Functions for Customizing Dialog Boxes

```
OSErr GXJobPrintDialog      (gxDialogResult *aDialogResult);
OSErr GXJobDefaultFormatDialog
                             (gxDialogResult *aDialogResult);
OSErr GXFormatDialog        (gxFormat aFormat, StringPtr title,
                             gxDialogResult *aDialogResult);
OSErr GXHandlePanelEvent    (gxPanelInfoRecord *aPanelInfoRecord,
                             gxPanelResult *aPanelResult);
OSErr GXFilterPanelEvent    (gxPanelInfoRecord *aPanelInfoRecord,
                             Boolean *returnImmed);
OSErr GXParsePageRange      (StringPtr fromString, StringPtr toString,
                             gxParsePageRangeResult *result);
```

Looping Through Format Objects

```
gxLoopStatus MyFormatFunction (gxFormat aFormat, void *refCon);
```

Dialog Box-Related Resources

The Panel Resource

```
type gxPrintPanelType {
    pstring[31]; /* the panel name */
    integer Script; /* script ID */
    fill word; /* reserve a long word for future use of
               international */
    fill word; /* reserve a long word for future use of
               international */
    integer; /* the icon ID */
    integer; /* the item list ID */
};
```

The Extended Item List Resource

```

type gxExtendedDITLType {
    integer = $$CountOf(xdtlarray) - 1;
    wide array xdtlarray {
        switch {
            case RadioButtons:
                key      integer = xdtlRadioButtons;
                literal  longint;    /* 4 byte id for storage in job
                                     object or format object */
                                longint;    /* numerical id for storage in
                                     job object or format object */
                integer;    /* offset in bytes into item */
                integer = $$CountOf(RadioButtonsArray) - 1;
                wide array RadioButtonsArray
                {
                    byte;    /* array of corresponding items*/
                };

            case CheckBox:
                key      integer = xdtlCheckBox;
                literal  longint;    /* 4-byte ID for storage in job
                                     object or format object */
                                longint;    /* numerical ID for storage in
                                     job object or format object */
                integer;    /* offset in bytes into item */
                byte;    /* corresponding ditl item */
                fill byte;

            case EditTextInteger:
                key      integer = xdtlEditTextInteger;
                literal  longint;    /* 4-byte ID for storage in
                                     job object or format object */
                                longint;    /* numerical ID for storage in
                                     job object or format object */
                integer;    /* offset in bytes into item */
                byte;    /* corresponding item list's item */
                byte;    /* 0 = dont select, 1 = select */
                pstring[15];/* low bound - nil means 'I
                                don't care' */
                pstring[15];/* high bound - nil means 'I
                                don't care' */

```

Page Formatting and Dialog Box Customization

```

case EditTextReal:
    key      integer = xdtlEditTextReal;
    literal  longint;    /* 4-byte ID for storage in job
                          object or format object */
                      longint;    /* numerical ID for storage in
                          job object or format object */
    integer;    /* offset in bytes into item */
    byte;      /* corresponding item list's item */
    byte;      /* 0 = don't select, 1 = select
    pstring[15];/* low bound - nil means 'I
                  don't care' */
    pstring[15];/* high bound - nil means 'I
                  don't care' */

case EditTextString:
    key      integer = xdtlEditTextString;
    literal  longint;    /* 4-byte ID for storage in job
                          object */
                      /* or format object */
    longint;    /* numerical ID for storage in
                          job object or format object */
    integer;    /* offset in bytes into item */
    byte;      /* corresponding item list's item */
    byte;      /* 0 = don't select, 1 = select */

case PopUp:
    key      integer = xdtlPopUp;
    literal  longint;    /* 4-byte ID for storage in job
                          object or format object */
    longint;    /* numerical ID for storage in
                          job object or format object */
    integer;    /* offset in bytes into item */
    byte;      /* corresponding item list's item */
    fill byte;

};
align word;
};
};

```

