

This chapter describes how you can develop printing extensions to modify or add to the printing features provided by QuickDraw GX. You need to read this chapter if you are developing printing extensions, including those that support a hardware device or modify the appearance of the pages that are printed by applications.

To use this chapter, you need to be familiar with how the printing features in QuickDraw GX work. *Inside Macintosh: QuickDraw GX Printing* describes these features and gives you an overview of the printing process. The chapter “Introduction to Printing Extensions and Drivers” in this book provides an overview of how QuickDraw GX provides printing and how you can override portions of its functionality in a printing extension or printer driver.

Printing extensions comprise a collection of resources. Before reading this chapter, you need some understanding of Macintosh resources and resource files. You can read about the Macintosh Resource Manager in *Inside Macintosh: More Macintosh Toolbox*.

Printing extensions make use of collections, which are managed by the Collection Manager, and are activated by messages, which are managed by the Message Manager. You need to know about the Collection Manager and Message Manager components of QuickDraw GX to understand how extensions work. Both of these managers are described in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

This chapter begins with an overview of printing extensions and then describes

- the kinds of tasks that you can use a printing extension to perform
- the development of a printing extension
- the resources that compose a printing extension
- some of the messages that you can override to develop a printing extension
- the interactions between printing extensions and QuickDraw GX

This chapter gives you the conceptual information that you need in the form of a guided tour through an extension that draws a background picture on each page of a document. For specific reference information about the messages that you override to write a printing extension, see the chapter “Printing Messages” in this book. For specific reference information about the resources that you need to include in your printing extension file, see the chapter “Printing Resources” in this book.

Note

The code samples presented in this chapter are taken from a recent version of the sample code. These samples are not complete and may have been slightly modified since printing. You can find the latest version of the code in the QuickDraw GX sample code. ♦

About Printing Extensions

Printing extensions are add-on software components that you can develop to extend the printing capabilities of QuickDraw GX. Printing extensions are used for tasks such as

supporting hardware additions and modifying the appearance of printed pages and allow you to provide these capabilities without having to implement an entire driver. For example, you could develop an extension that allows users to control a sheet feeder on a printer or that adds a confidential stamp to every page produced by an application.

Just as Macintosh system software extends its functionality with system extensions and control panels, QuickDraw GX allows you to extend the functionality of printing by providing printing extensions. You can

- add panels to the QuickDraw GX print dialog boxes
- obtain additional information from users to control options within your extension
- modify the manner in which each page, or a whole document, is printed
- drive hardware extensions to printing devices

This added functionality is completely independent of the mainstream execution of an applications, and users can activate an extension at the time they run an application. Active printing extensions reside in the Extensions folder inside the System Folder, along with system extensions and control panels. Unlike system extensions and control panels, which run at system startup time, printing extensions run at print time. Any number of printing extensions can be active for a given print job.

Application developers can create applications that initiate printing functions without knowing whether printing extensions will run with the application. When a printing extension is active, QuickDraw GX adds the activity of the extension to that of the requested printing function. If no extensions are active, QuickDraw GX just performs the requested printing function. QuickDraw GX takes care of locating and activating the printing extensions at the appropriate time. All you have to do when you develop the extension is to provide QuickDraw GX with information about how and when you want the extension used.

Since printing extensions run independently of applications, a single extension can run with a number of different applications. Thus, for example, you can write an extension that adds a special stamp to a printed page, and users can activate it whenever they want that stamp added to their printed documents.

Printing extensions also allow you to write a single piece of code to handle hardware add-on devices rather than rewriting a driver to do it. By doing this, you provide users with the ability to support the hardware add-on for multiple printers. Your extension can add a panel to one of the print dialog boxes to extend control of the device to the user. For example, the same extension can drive a sheet feeder device that you can attach to a number of different printers.

Printing Extension Tasks

Printing extensions can affect printing at well-defined points during the printing process. QuickDraw GX searches for active extensions and sends messages to them to allow you to extend printing capabilities. You can

Printing Extensions

- add panels to the Page Setup, Custom Page Setup, and Print dialog boxes and monitor the user's activity in those panels
- modify printed pages during the spooling phase of printing
- modify printed pages during the imaging phase of printing
- modify the manner in which document pages are sent to the printer during the device communications phase of printing

Each of these extension capabilities is described in the sections that follow.

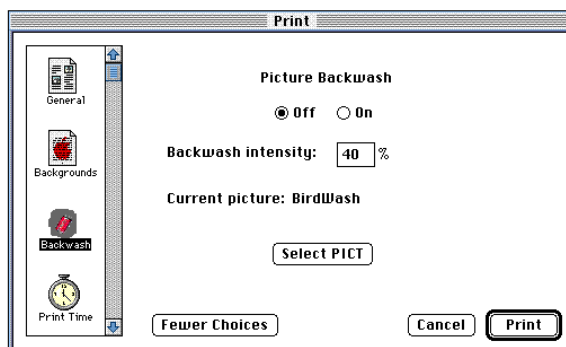
Printing phases are described in the chapter “Introduction to Printing Extensions and Drivers” in this book. The dialog boxes used during the printing process are described in detail in *Inside Macintosh: QuickDraw GX Printing*. The chapter “Printing Messages” in this book describes each of the messages that QuickDraw GX sends during the printing process.

Adding Panels to Print Dialog Boxes

QuickDraw GX allows applications, extensions, and drivers to add panels to the print dialog boxes. Adding a panel and monitoring the user's selections in the dialog box is described in *Inside Macintosh: QuickDraw GX Printing*. While the user manipulates the panel, QuickDraw GX sends event messages (the `GXFilterPanelEvent` and the `GXHandlePanelEvent` messages) to your extension. It also sends messages before closing a panel and notifies you whether the user confirmed or canceled the dialog box. The `GXFilterPanelEvent` message is described on page 4-86 and the `GXHandlePanelEvent` message is described on page 4-85 in the chapter “Printing Messages.”

You can add panels for users to control the capabilities that your printing extension provides. For example, the background picture printing extension adds a panel to the expanded Print dialog box that is displayed when the user clicks the More Choices button in the Print dialog box. This panel allows the user to specify which picture file to use as the background picture on printed pages, as shown in Figure 2-1.

Figure 2-1 The background picture panel displayed in the Print dialog box



Modifying a Page During Spooling

You can develop a printing extension to make device-independent changes to image data during the spooling phase of printing. QuickDraw GX sends a message to the extension just before each page in a document is spooled to disk. Your printing extension can take this opportunity to make any changes to the page that it wishes. The changes are then incorporated into the final version of the page that is saved in the spool file.

For example, you might develop an encryption printing extension that is used during spooling to encode the printing data as it is spooled to disk and decode it as it is read for printing. This extension allows security-conscious organizations to print sensitive documents and ensure that they are secure during the time they reside in spooler files. It also allows the spooled files to be distributed over a network with confidence that the contents are hidden during transmission time.

Modifying a Page During Imaging

You can develop a printing extension to modify a page in either a device-independent or device-dependent manner during the imaging phase of printing. QuickDraw GX sends messages to the extension just after opening the spool file for imaging and just before closing the spool file at the completion of imaging. Examples of printing extensions that modify the page are:

- A confidential stamp printing extension that stamps the word “Confidential” across each page of a document as it is printed. You can set up a shared confidential printing station with this extension attached to it, and any document printed to this printing station will have the confidential stamp added to its pages.
- A background picture printing extension that gives a user a choice of backgrounds to print against. For example, a user could use this extension to add color ramps to a document being printed as slides for a presentation. Or a user could specify a picture file to use as a background picture, as shown in Figure 2-2 on page 2-8.
- A thumbnail printing extension that reduces the page size so that a number of logical pages can be printed together on one physical page. This is useful for proofing and creating story boards.

Modifying the Device Communications Process

You can develop a printing extension to modify the manner in which each printed page is sent to the printer. For example, you can develop an extension that creates a PostScript file copy of each document that is printed. This extension overrides the printing message that sends data to the printer (the `GXDumpBuffer` message, which is described on page 4-142 in the chapter “Printing Messages”) and copies the buffer to a file.

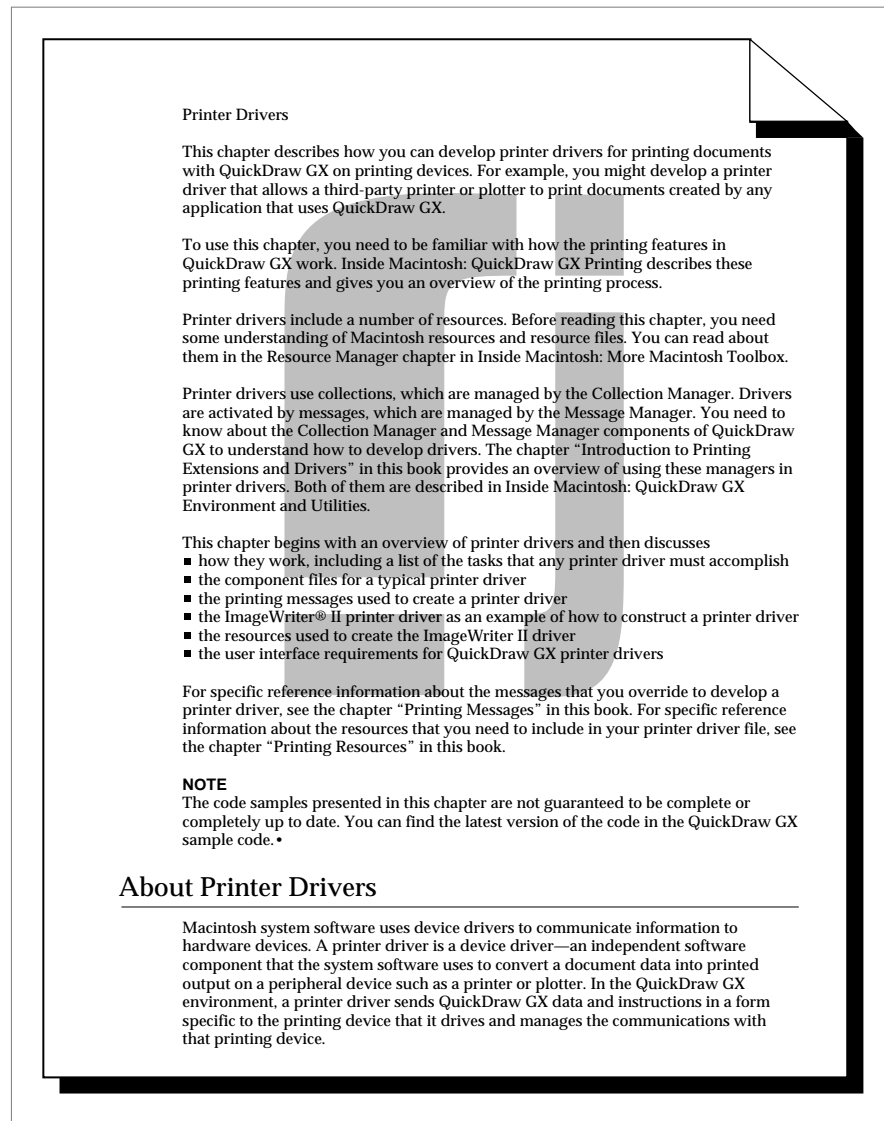
Developing a Printing Extension

The composition of your printing extension depends on the capabilities that you are adding to QuickDraw GX printing. Each printing extension must at least include a resource file, an assembly-language jump table, one or more files of code to implement the functions that your extension perform. The background picture printing extension is typical; it consists of four files, as shown in Table 2-1. The rest of this chapter describes the contents of each of these files. The complete files are found in the QuickDraw GX sample code.

Table 2-1 Files used to implement the background picture printing extension

Filename	Contents
backwash.a	The assembly-language jump table that QuickDraw GX uses to locate the message overrides for the background picture printing extension
backwash.h	The C language header file that includes all of the other needed header files, defines the constants and data types used by the program, and references external items
backwash.c	The C language code that implements the message overrides for the background picture printing extension
backwash.r	The resources for the background picture printing extension

The background picture printing extension draws a background picture on each page as the page is despooled, just before it is imaged on the output device. A page printed while the background picture printing extension is active can have any picture printed on it. An example is shown in Figure 2-2.

Figure 2-2 A page printed while the background picture printing extension is active

The background picture printing extension also adds a panel to the Print dialog box. The panel allows the user to enable or disable the extension, to select which background picture to use, and to set an intensity level for drawing the picture on the page. The Print dialog box with the background picture panel added is shown in Figure 2-1 on page 2-5.

The Jump Table

You need to tell QuickDraw GX where (in which segment and at what offset from the beginning of that segment) to find the code for each printing message that you are overriding. You do this by defining an assembly-language jump table. The contents of the file `backwash.a`, which defines the jump table for the background picture printing extension, are shown in the QuickDraw GX sample code.

The jump table links the appropriate override function into your code and provides a jump statement to invoke that function. The override resource tells QuickDraw GX where to look (at which offset) in the jump table to find the jump statement for a specific printing message name. In this way, QuickDraw GX knows which of your override functions to call to respond to the messages in which you are interested.

You define a jump table with one jump (`JMP`) statement for each message that you override. You also define an override resource that specifies the offset in that jump table for each message. The jump table and override resource must be coordinated. QuickDraw GX dispatches a printing message to your extension by jumping to the routine whose location is specified in the appropriate location in the jump table. The override resource is described in the section “The Override (‘over’) Resource” beginning on page 6-13 in the chapter “Printing Resources.”

For example, the background picture printing extension overrides several printing messages, as shown in Table 2-2.

Table 2-2 Printing messages overridden by the background picture printing extension

Message	Why you override
GXInitialize	To set up the environment so that the extension can use predefined global values
GXShutDown	To free the storage that was allocated by the GXInitialize message override
GXJobPrintDialog	To add a panel to the Print dialog box so the user can select the background picture filename and intensity
GXHandlePanelEvent	To handle events that occur in the panel
GXCreateSpoolFile	To add the background picture to the spool file
GXDespoolPage	To draw the background picture on each page as it is despoiled
GXCloseSpoolFile	To free the storage allocated for the background picture shape

The override resource for the background picture printing extension has to include an entry for each of these printing messages, and the jump table has to include a jump statement for each. Listing 2-1 shows the override resource definition from the `backwash.r` file. You can read about each of the printing messages in the chapter “Printing Messages” in this book.

Listing 2-1 The override resource for the background picture printing extension

```
resource gxOverrideType (gxExtensionUniversalOverrideID,sysHeap,
                        purgeable)
{
    {
        gxInitialize,      0, 4,
        gxShutDown,        0, 8,
        gxJobPrintDialog,  ' 0, 12,
        gxHandlePanelEvent, 0, 16,
        gxCreateSpoolFile,  0, 20,
        gxDespoolPage,      0, 24,
        gxCloseSpoolFile,   0, 28
    };
};
```

The name of each message is followed by the ID of the extension's code segment in which its code resides and the byte offset of its jump statement in the jump table. QuickDraw GX reserves the first 4 bytes for its own use, which makes 4 the first offset that you can use. These 4 bytes are used by QuickDraw GX to maintain an owner count and must be set to 0 in the jump table.

You implement the jump table as an assembly-language program that contains a jump statement for each override function. You need to list the jump statements with exactly the same offsets as you listed for the message names in your override resource; otherwise, QuickDraw GX will invoke the wrong function in response to a message. Listing 2-2 shows the jump table for the background picture printing extension.

Listing 2-2 The jump table for the background picture printing extension

```
EXPORT EntryPoint
IMPORT BWInitialize
IMPORT BWShutDown
IMPORT BWJobPrintDialog
IMPORT BWHandlePanelEvent
IMPORT BWCreateSpoolFile
IMPORT BWDespoolPage
IMPORT BWCloseSpoolFile

EntryPoint PROC                ; main entry point

    DC.L    0                  ; used by QuickDraw GX

    JMP     BWInitialize        ; override for GXInitialize
    JMP     BWShutDown          ; override for GXShutDown
```


Printing Extensions

```

JMP      BWJobPrintDialog      ; override for GXJobPrintDialog
JMP      BWHandlePanelEvent    ; override for GXHandlePanelEvent
JMP      BWCreateSpoolFile     ; override for GXCreateSpoolFile
JMP      BWDespoolPage         ; override for GXDespoolPage
JMP      BWCloseSpoolFile      ; override for GXCcloseSpoolFile

ENDPROC

END

```

Note

The code shown in Listing 2-2 is for the MPW environment. If you are programming in a different development environment, you might need to use different assembler directives. You must, however, be certain to include the initial 4 bytes (set to 0) and each `JMP` statement. ♦

The `EXPORT` statement at the beginning makes the jump table public. The `IMPORT` statements make it possible for your assembly-language program to reference the C language functions performing your message overrides and to provide correct linkage to those functions.

The name of each override function provided by the background picture printing extension is prefixed with `BW` to differentiate it from other overrides of the same message. This means that the extension's override of the `GXInitialize` message is named `BWInitialize`, its override of the `GXDespoolPage` message is named `BWDespoolPage`, and so on.

Because QuickDraw GX uses the first 4 bytes in the jump table to store the owner count value, the first statement (`DC.L`) must be included. These bytes must all have the value 0 in them.

You must include one `JMP` statement for each message that you override. You can choose to intersperse the `IMPORT` and `JMP` statements as shown in Listing 3-2 on page 3-15 in the chapter “Printer Drivers,” or you can place all of the `IMPORT` statements together, followed by all of the `JMP` statements, as is shown in Listing 2-2.

IMPORTANT

Always coordinate the entries in your override resources with the entries in your jump table. If they are not aligned, the wrong code will be executed to override a message. The offset that you specify in the resource for each message must match the offset of the corresponding override function in your jump table. You must also include 4 bytes with zero values at the beginning of your jump table. ▲

The Printing Message Overrides

The code that makes up your printing extension is a set of functions that override, either partially or totally, some of the printing messages that QuickDraw GX sends during the process of printing a document. The contents of the file `backwash.c`, which contains

Printing Extensions

the source code for the message overrides in the background picture printing extension, are shown in the QuickDraw GX sample code.

QuickDraw GX provides a default implementation of each printing message that it sends. You can augment (partially override) some printing messages, and you can replace (totally override) others. For each printing message, QuickDraw GX provides one of three kinds of default implementations, as shown in Table 2-3.

Table 2-3 Implementation types for QuickDraw GX printing messages

Message type	Default implementation
Empty	Does nothing, so you can provide an override of this message to insert functionality into the printing process. You can choose to partially or totally override this message.
Can be partially overridden	Provides necessary functionality, which you augment by partially overriding it. You must forward this message to other message handlers.
Can be totally overridden	Provides basic functionality, which you can either partially or totally override.

Whenever you override a QuickDraw GX printing message, you must be certain that the declaration of your override function matches the declaration of the message. This means that the type of function return and the type of each parameter must match the types in the message declaration. The chapter “Printing Messages” shows the declaration of each printing messages.

When you partially override a printing message, you add some functionality to that provided by other message handlers, including QuickDraw GX (through its default implementation), the printer driver, and other printing extensions. You forward the message to these other handlers, as described in the section “Forwarding Messages” on page 2-13.

When you totally override a message, you replace any functionality that is provided by other message handlers, including the default implementation. Your total message override does not forward the message to the other handlers. This means that you must be sure to replace the functionality that is provided by the default implementation.

Although the default implementation of a message might be empty, other printing message handlers (the printer driver and other printing extensions) can also override messages, so you need to carefully consider whether or not to create a total override of a message.

Whether or not you can totally override a message and when you need to forward a message in your implementation of a partial override is specific to each message. The reference section “Printing Messages Reference” beginning on page 4-9 in the chapter “Printing Messages” provides this information for each message.

Choosing the Messages to Override

Which messages you need to override for your printing extension depends entirely on what you want your extension to do. There are not any messages that you are required to override. Some extensions override many messages to provide their operations, and many extensions are created by overriding only a few messages. Refer to the chapter “Printing Messages” in this book for complete information about the available messages.

Forwarding Messages

Your printing extension can forward a message in its implementation of a partial override. If you are totally overriding a message, you do not forward it to other message handlers. You can forward the message either before or after performing your own actions. To forward a message to the next message handler in the message chain, use a statement with the following format:

```
anErr = Forward_MessageName(arguments);
```

For example, the background picture printing extension overrides the `GXJobPrintDialog` message to add a panel to the Print dialog box. This message must be forwarded so that the default implementation can build the default dialog box and any other message handlers can add their panels to the dialog box. Listing 2-3 shows the override of the `GXJobPrintDialog` message from the background picture extension.

Listing 2-3 Forwarding the `GXJobPrintDialog` message

```
OSErr BWJobPrintDialog (gxDialogResult *dlogResult)
{
    OSErr err;

    err = SetupPrintPanel();
    if (!err)
        err = Forward_GXJobPrintDialog(dlogResult);
    return err;
}
```

This override function calls a local function named `SetupPrintPanel` to add items to the Print dialog box that are used for controlling the background picture extension. If that function succeeds, this version forwards the message to the next message handler, which can add its own panel. The `GXJobPrintDialog` message is described on page 4-84 in the chapter “Printing Messages.”

Sending Messages

Your printing extension can also send a printing message to other handlers in the message chain. When you send a message, QuickDraw GX receives it and then sends it to the first message handler in the chain. To send a message, use a statement with this format:

```
anErr = Send_GXMessageName(arguments);
```

For example, to send the `GXBufferData` message, which is described on page 4-139 in the chapter “Printing Messages,” use the following statement:

```
anErr = Send_GXBufferData(gxDialogResult *);
```

For more information on sending messages, refer to the chapter “Message Manager” in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Handling Exceptions in Your Message Overrides

The code samples presented in this chapter make use of an exception-handling strategy that simplifies the job of testing for error conditions after each function call. This strategy uses three C macros, each of which branches to an error-handling statement in response to a condition.

The first macro, `nrequire`, branches to a label when the value of its `condition` argument is anything other than 0. This macro is commonly used to test the result code from a function call and branch if the function returned an error (any value other than `noErr`, which is 0). The syntax of the `nrequire` macro is

```
nrequire(condition, location);
```

An example of using the `nrequire` macro is shown in Listing 2-4.

Listing 2-4 Using the `nrequire` macro for exception handling

```
OSErr      err;
long       count;
short      pictRefNum;
PicHandle  backwashPict = nil;

err = FSpOpenDF(opFSSpec, fsCurPerm, &pictRefNum);
nrequire(err, CouldNotOpenFile);

err = GetEOF(pictRefNum, &count);
nrequire(err, CouldNotGetEOF);
count -= 512;
```

Printing Extensions

```

    err = SetFPos(pictRefNum, fsFromStart, 512);
    nrequire(err, CouldNotSetFilePos);

    ...

CouldNotGetEof:
CouldNotSetFilePos:
    FSClose(pictRefNum);

CouldNotOpenFile:
    return(backwashPict);
}

```

The function in Listing 2-4 uses the `nrequire` macro instead of using an `if` statement to test the value of `err` after each call. The `nrequire` macro performs a branch to the specified label if the value of `err` is anything other than `noErr`. Each label that is referenced by the `nrequire` macro leads to a call that cleans up after the error.

A variation of the `nrequire` macro that is used in some functions is the `require` macro. This macro is exactly the same as `nrequire` except that it branches when its condition argument is 0 (whereas `nrequire` branches on anything other than 0). The syntax of the `require` macro is

```
require(condition, location);
```

A final variation of the `nrequire` macro that is used in some of the functions of the background picture printing extension is the `nrequire_action` macro. This macro works the same way as `nrequire` and additionally performs an action before branching. The syntax of the `nrequire_action` macro is

```
nrequire_action(condition, location, action);
```

The action is performed only if the value of the `condition` argument is anything other than 0. The action is performed before branching to the specified label. An example of the `nrequire_action` macro is shown in Listing 2-5.

Listing 2-5 Using the `nrequire_action` macro for exception handling

```

grErr = GetJobCollectionItem(&backwashConfig, nil,
                           kBackwashCollectionType, kBackwashSettingsID);
nrequire_action(!grErr && !backwashConfig.addBackwash &&
               backwashConfig.haveFileInfo,
               NotAddingBackwash, grErr = noErr);

```

The `nrequire_action` call in this example assigns the value `noErr` to the `grErr` variable before branching to the `NotAddingBackwash` label.

Implementing the Background Picture Extension Functions

This section describes the functions of the background picture printing extension. This extension needs to accomplish the following tasks:

1. Initialize the environment.
2. Add a panel to the Print dialog box so that the user can make several choices: whether to enable or disable this extension, which picture to use as the background picture, and at what intensity level to draw the picture on the page.
3. Respond to events that occur in the panel, such as a mouse click or keypress.
4. Store the background picture in the spool file along with the document that is being spooled.
5. Draw the selected background picture on each page of a document that is being despoiled.
6. Deallocate the storage allocated for the background picture shape that was spooled.
7. Shut down the environment at the end of the program.

Initializing the Extension Environment

The background picture printing extension performs very simple initialization. Its override of the `GXInitialize` message, `BWInitialize`, allocates a globals world so that it has the global data needed for a printing extension. The `GXInitialize` message is described on page 4-43 in the chapter “Printing Messages.” The code for the `BWInitialize` function is shown in Listing 2-6.

Listing 2-6 The `BWInitialize` override function

```
OSErr BWInitialize()
{
    OSErr err = noErr;

    err = NewMessageGlobals(A5Size(), A5Init);
    if (!err) err = InitGlobalData();
    return err;
}
```

The `BWInitialize` function first sets up an A5 world, which you must do in your extension if you are going to use global data. Setting up an A5 world means setting the A5 register to reference your global data. The functions `A5Size` and `A5Init` are supplied for creating an A5 world.

`BWInitialize` calls a local function named `InitGlobalData` to initialize any global variables that are defined by the background picture printing extension. This function is shown in Listing 2-7.

IMPORTANT

You must perform the actual initialization of your global variables in a function that you call from your override of the `GXInitialize` message. This is because some compilers optimize code in a way that can invalidate the A5 register. This is why the `BWInitialize` override function calls the `InitGlobalData` function to initialize the globals. ▲

Listing 2-7 The `InitGlobalData` function

```
OSErr InitGlobalData()
{
    gBackwashShape = nil;
    return noErr;
}
```

This function initializes the `gBackwashShape` variable to `nil` to indicate that it is currently unused.

Adding a Background Picture Panel to the Print Dialog Box

The background picture printing extension adds a panel to the Print dialog box. The placeholders for this panel are defined in the `backwash.r` resource file; however, several of the values need to be filled in at run time. In the background picture printing extension, the override of the `GXJobPrintDialog` message, `BWJobPrintDialog`, performs this operation by calling a local function, `SetupPrintPanel`, and then forwarding the `GXJobPrintDialog` message to the rest of the message handlers. The `GXJobPrintDialog` message is described on page 4-84 in the chapter “Printing Messages.” The `BWJobPrintDialog` function is shown in Listing 2-8.

Listing 2-8 The `BWJobPrintDialog` override function

```
OSErr BWJobPrintDialog(gxDialogResult *dlogResult)
{
    OSErr err;

    err = SetupPrintPanel();

    if (!err)
        err = Forward_GXJobPrintDialog(dlogResult);

    return err;
}
```

Printing Extensions

The local function `SetupPrintPanel` performs the work of setting up the panel that the background picture extension adds to the Print dialog box. It uses the Collection Manager to store and access a `BackwashCollection` structure. The `BackwashCollection` structure is defined as shown in Listing 2-9. The Collection Manager is described in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Listing 2-9 The `BackwashCollection` structure

```
struct BackwashCollection {
    long            intensity;
    unsigned char   addBackwash;
    Boolean         haveFileInfo;
    FSSpec          fileInfo;
};

typedef struct BackwashCollection BackwashCollection;
```

The fields of this structure are used as follows:

Field descriptions

<code>intensity</code>	The intensity level (a percentage value) that the user has selected for drawing the background picture on the page.
<code>addBackwash</code>	A value that indicates whether the user has enabled the background picture printing extension.
<code>haveFileInfo</code>	A Boolean value that indicates whether the user has entered a picture filename into the panel.
<code>fileInfo</code>	The file system specification of the picture file that the user chose to use as the background picture.

The `SetupPrintPanel` function first attempts to access the background picture collection item from the job collection, which is the collection of items maintained by the Collection Manager for the print job that is printing. If this is the first time that the background picture extension has been accessed, the item is not found, and `SetupPrintPanel` creates a new item with default values and adds it to the collection. Finally, `SetupPrintPanel` sets up the panel information for the Dialog Manager and calls the `GXSetupDialogPanel` function to actually add the panel to the dialog box. The `GXSetupDialogPanel` function is described on page 5-28 in the chapter “Printing Functions for Message Overrides.” The code for the `SetupPrintPanel` function is shown in Listing 2-10.

Listing 2-10 The `SetupPrintPanel` function

```
OSErr SetupPrintPanel()
{
    OSErr          noErr;
```


Printing Extensions

```

    gxPanelSetupRecord    panelSetupRec;
    BackwashCollection    backwashConfig;

/*
    Get the job collection and then try to find the
    backwash collection item in there.
*/

    jobCollection = GXGetJobCollection(GXGetJob());
    /* call local function to get settings */
    err = GetJobCollectionItem(&backwashConfig, nil,
                              kBackwashCollectionType, kBackwashSettingsID);

/*
    If the collection item doesn't yet exist, create a new
    item, set it up with default values, and add it to the
    job collection
*/

    if (err == collectionItemNotFoundErr)
    {
        backwashConfig.addBackwash = kDontAddBackwash;
        backwashConfig.haveFileInfo = false;
        backwashConfig.intensity = kDefaultIntensity;
        strcpy(&backwashConfig.fileInfo.name[1], "none");
        backwashConfig.fileInfo.name[0] =
            strlen(&backwashConfig.fileInfo.name[1]);

        /* call local function to store settings */
        err = StoreJobCollectionItem(&backwashConfig,
                                     sizeof(BackwashCollection),
                                     kBackwashCollectionType,
                                     kBackwashSettingsID, true);
    }

    nrequire(err, GetSettings_Failed);

/*
    Set up the panel: store the ID of the panel resource to use,
    the resource file in which it is located, and the type of
    panel that is being stored.
*/

    panelSetupRec.panelResId= r_BackwashPanel;    /* resource ID */

```

Printing Extensions

```

/* resource file */
panelSetupRec.resourceRefNum = GXGetMessageHandlerResFile();
panelSetupRec.refCon = 0;          /* not used*/
panelSetupRec.panelKind = gxExtensionPanel; /* panel type */

err = GXSetupDialogPanel(&panelSetupRec);

GetSettings_Failed:

    return err;
}

```

SetupPrintPanel returns an error code of type `OSErr`, which is the same type of function result that is returned by all of the functions that it calls. SetupPrintPanel also uses the error code to detect when it needs to initially add its own item to the job collection.

▲ **WARNING**

The size of the items in the job collection is subject to change as QuickDraw GX evolves. For that reason, it is important for you to specify an expected size for each item in your calls to the Collection Manager, rather than specifying `nil`, which tells the Collection Manager to copy the entire object no matter what its size is. The size that you specify must match the size of the data structure into which the item is being copied. ▲

Handling an Event in the Background Picture Panel

When a user event occurs in the background picture panel, the background picture printing extension needs to examine the event and determine whether or not to handle it. QuickDraw GX sends the `GXHandlePanelEvent` message whenever an event occurs in a printing panel. The `GXHandlePanelEvent` message is described on page 4-85 in the chapter “Printing Messages.” Included with the message is a structure of type `gxPanelInfoRecord`, which is declared as shown in Listing 2-11.

Listing 2-11 The `gxPanelInfoRecord` structure

```

struct gxPanelInfoRecord {
    gxPanelEvent panelEvt; /* the event */
    short panelResId;      /* resource ID of current 'ppnl' res */
    DialogPtr pDlg;        /* pointer to dialog box */
    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 occurs after

```

Printing Extensions

```
                                this event is processed */
short errorStringId;          /* ID of error string */
gxFormat theFormat;           /* the current format */
void *refCon;                 /* refCon from gxPanelSetupRecord */
};
```

```
typedef struct gxPanelInfoRecord gxPanelInfoRecord;
```

The fields of this structure are described in the section “The Panel Information Structure” on page 4-35 in the chapter “Printing Messages.”

Panel events and the way that they are processed are described in *Inside Macintosh: QuickDraw GX Printing*.

In the background picture printing extension, the override of the `GXHandlePanelEvent` message, `BWHandlePanelEvent`, responds to the panel events that are shown in Table 2-4. The list of possible panel events that you can respond to is given in the section “Panel Events” on page 4-36 in the chapter “Printing Messages.”

Table 2-4 Panel events handled by the background picture printing extension

Constant	Explanation
<code>gxPanelOpenEvt</code>	The panel is about to open. It needs to be initialized and drawn. The background picture extension responds to this event by initializing the current filename that is displayed in the panel.
<code>gxPanelHitEvt</code>	The user has selected an item in the panel. The background picture extension responds when the user selects the “Select PICT” panel item by replacing the file specification in the job collection with the filename that the user selects.
<code>gxPanelActivateEvt</code>	The dialog box window in which the panel resides has just been activated. The background picture extension responds to this event by activating the picture intensity field.
<code>gxPanelDeactivateEvt</code>	The dialog box window in which the panel resides is about to be deactivated. The background picture extension responds to this event by deactivating the picture intensity field.
<code>gxPanelIconFocusEvt</code>	The focus has changed from the panel to the icon list. The background picture extension responds to this event by deactivating the picture intensity field.
<code>gxPanelPanelFocusEvt</code>	The focus has changed from the icon list to the panel. The background picture extension responds to this event by activating the picture intensity field.

Printing Extensions

The `BWHandlePanelEvent` function, which is shown in Listing 2-12, operates as a switch statement that selects the events of interest (those listed in Table 2-4) and provides code to handle each. The `gxPanelOpenEvt` case is handled by calling a local function, `OpenBackwashPanel`, which is shown in Listing 2-13 on page 2-24.

Listing 2-12 The `BWHandlePanelEvent` override function

```

OSErr BWHandlePanelEvent(gxPanelInfoRecord *panelInfo)
{
    OSErr          err = noErr;
    GrafPtr        oldPort;
    DialogPtr       pDlg;
    StandardFileReply reply;
    SFTypeList      typeList;
    BackwashCollection backwashConfig;
    Handle          hItem;
    Rect            itemRect;
    short           itemType;

    pDlg = panelInfo->pDlg;
    GetPort(&oldPort);
    SetPort(pDlg);

    switch (panelInfo->panelEvt) /* select events of interest */
    {
        /* if the panel is opening, initialize it */
        case gxPanelOpenEvt:
            OpenBackwashPanel(pDlg, panelInfo->itemCount);
            break;

        /*
         * If the user clicks the Select Picture button, prompt for the
         * name of a file to load. If the user selects one, access the
         * collection item, move the user's file specification to the
         * collection, and replace the old collection item with the
         * modified version.
         */
        case gxPanelHitEvt:
            if (panelInfo->itemHit == (panelInfo->itemCount +
                                     d_SelectPicture))
            {
                typeList[0] = 'PICT';
                StandardGetFile(nil, 1, typeList, &reply);
            }
    }
}

```

Printing Extensions

```

require(replay.sfGood, UserCancelledFileDialog);

err = GetJobCollectionItem(&backwashConfig, nil,
                          kBackwashCollectionType, kBackwashSettingsID);

nrequire(err, GetSettings_Failed);

backwashConfig.haveFileInfo = true;
BlockMove(&reply.sfFile,
          &backwashConfig.fileInfo, sizeof(FSSpec));

/* replace old collection item with updated version */
err = StoreJobCollectionItem(&backwashConfig,
                             sizeof(BackwashCollection),
                             kBackwashCollectionType,
                             kBackwashSettingsID, false);
nrequire(err, StoreSettings_Failed);

/* update the filename dialog item */
GetDItem(panelInfo->pDlg,
          panelInfo->itemCount + d_FileNameItem,
          &itemType, &hItem, &itemRect);
SetIText(hItem, &backwashConfig.fileInfo.name);
}
break;

/*
If the panel is activating or deactivating, or if the focus
(the section of the dialog box that is active) is changed,
either activate or deactivate the picture intensity field.
*/
case gxPanelActivateEvt:
case gxPanelDeactivateEvt:
case gxPanelIconFocusEvt:
case gxPanelPanelFocusEvt:
    if (((DialogPeek) pDlg)->editField + 1) ==
        (panelInfo->itemCount + d_intensity))
    {
        if ((panelInfo->panelEvt == gxPanelPanelFocusEvt)
            TEEActivate(((DialogPeek) pDlg)->textH);
        else
            TEDeactivate(((DialogPeek) pDlg)->textH);
    }
}

```

Printing Extensions

```

        break;
    }

    UserCancelledFileDialog:
    GetSettings_Failed:
    StoreSettings_Failed:
        SetPort(oldPort);
        return err;
}

```

The `OpenBackwashPanel` function in Listing 2-13 is a local function called by the `BWHandlePanelEvent` function to initialize the background picture panel when the user opens it. This function performs any initialization of the panel that cannot be performed by 'xdt1' resource specifications, which are described in *Inside Macintosh: QuickDraw GX Printing*.

Listing 2-13 The `OpenBackwashPanel` function

```

/*
    OpenBackwashPanel handles non-'xdt1' item initialization when
    the panel is opened. Note that the items are offset from the
    the value of itemCount, which means that item #5 in the panel
    is accessed by passing the value itemCount+5.
*/

void OpenBackwashPanel(DialogPtr pDlg, short itemCount)
{
    BackwashCollection    backwashConfig;
    Handle                hItem;
    Rect                  itemRect;
    short                 itemType;

    /*
        Initialize the current filename displayed, based on the
        settings in the backwash collection item.
    */

    GetJobCollectionItem(&backwashConfig, nil,
                        kBackwashCollectionType, kBackwashSettingsID);

    GetDItem(pDlg, itemCount + d_FileNameItem, &itemType,

```

Printing Extensions

```

        &hItem, &itemRect);
    SetIText(hItem, &backwashConfig.fileInfo.name);
}

```

The `OpenBackwashPanel` function initializes the dialog box panel by filling in the filename that is stored in its configuration information. It operates by retrieving the configuration information from the job collection and storing the filename from the configuration into the filename item in the panel.

Storing the Background Picture in the Spool File

The background picture printing extension draws the background picture on each page. To do this, it first stores the background picture as a resource in the spool file. It then accesses that resource and applies the picture to each page as the page is being despoiled.

To store the picture in the spool file, the background picture printing extension overrides the `GXCreateSpoolFile` message, which QuickDraw GX sends at the start of spooling a document. The `GXCreateSpoolFile` message is described on page 4-68 in the chapter “Printing Messages.” In the background picture printing extension, the override of `GXCreateSpoolFile`, `BWCreateSpoolFile`, is shown in Listing 2-14.

Listing 2-14 The `BWCreateSpoolFile` override function

```

OSErr BWCreateSpoolFile(FSSpecPtr anFSSpec, long createOptions,
                        gxSpoolFile *theSpoolFile)
{
    gxGraphicsError    grErr;
    BackwashCollection backwashConfig;

    /* forward the message so that the spool file is created */
    grErr = (gxGraphicsError) Forward_GXCreateSpoolFile(anFSSpec,
                                                         createOptions, theSpoolFile);
    nrequire(grErr, ForwardMessage_Failed);
    /*
     * Get the collection item and see if extension is enabled and
     * picture file info has been entered. If so, add the background
     * picture to the spool file.
     */

    grErr = (gxGraphicsError) GetJobCollectionItem(&backwashConfig,
                                                    nil, kBackwashCollectionType, kBackwashSettingsID);

    if (!grErr && backwashConfig.addBackwash &&
        backwashConfig.haveFileInfo)

```

Printing Extensions

```

        grErr = AddBackwash(&backwashConfig.fileInfo,
                           (short) backwashConfig.intensity, *theSpoolFile);
    else
        if (grErr == collectionItemNotFoundErr)
            grErr = noErr;

ForwardMessage_Failed:
    return (OSErr) grErr;
}

```

The `BWCreateSpoolFile` function first forwards the `GXCreateSpoolFile` message to allow QuickDraw GX to use the default implementation (or other message handlers to use their implementations) to create the spool file. `BWCreateSpoolFile` then calls the local function `AddBackwash` to read the background picture file and store it in the spool file. The `AddBackwash` function is shown in Listing 2-15.

Listing 2-15 The `AddBackwash` function

```

gxGraphicsError AddBackwash(FSSpecPtr fileInfo,
                           short theIntensity,
                           gxSpoolFile theSpoolFile)
{
    gxGraphicsError    grErr = noErr;
    Rect               pictBounds;
    PicHandle          backwashPict;
                        /* use {1, 1} to mean don't stretch */
    Point              patStretchPoint = {1,1};
    Handle              gxShapeHdl = nil;
    gxShape             gxPictShape;
    short              resAttribs;

    /* load the picture and then convert it to a GX shape */
    backwashPict = LoadAPict(fileInfo);
    require_action(backwashPict, CouldNotLoadPICT,
                  grErr = nilHandleErr);

    pictBounds = (*backwashPict)->picFrame;
    gxPictShape = GXNewShape(gxPictureType); /* picture shape */
    nrequire_action(GXGetGraphicsError(&grErr),
                  CouldNotCreateShape, KillPicture(backwashPict));

    GXConvertPICTToShape(backwashPict, gxDefaultOptionsTranslation,
                        &pictBounds, &pictBounds, patStretchPoint,
                        gxPictShape, nil);
}

```


Printing Extensions

```

KillPicture(backwashPict);          /* get rid of original */
nrequire_action(GXGetGraphicsError(&grErr),
                CouldNotConvertShape, GXDisposeShape(gxPictShape));

/* lighten the picture shape as specified by user */
grErr = SetShapeIntensity(gxPictShape, theIntensity);
nrequire_action(grErr, CouldNotSetShapeIntensity,
                GXDisposeShape(gxPictShape));

/* flatten picture shape into handle */
gxShapeHdl = ShapeToHandle(gxPictShape);
grErr = (gxGraphicsError) MemError();
GXDisposeShape(gxPictShape);
nrequire(grErr, CouldNotFlattenShape);
/*
Add the picture shape as a resource in the spool file. After
adding it, mark the resource as "sysHeap, purgeable", so that
it won't be loaded into PrinterShare's heap, which is small.
*/
grErr = Send_GXSpoolResource(theSpoolFile, gxShapeHdl,
                             kBackwashCollectionType, r_BackwashPICTID);
nrequire_action(grErr, CouldNotAddShape,
                DisposHandle(gxShapeHdl));

resAttribs = GetResAttrs(gxShapeHdl);
SetResAttrs(gxShapeHdl,
            resAttribs | resSysHeap | resPurgeable);
ChangedResource(gxShapeHdl);
WriteResource(gxShapeHdl);

/* release the resource-DO NOT call DisposHandle on it */
ReleaseResource(gxShapeHdl);

CouldNotLoadPICT:
CouldNotCreateShape:
CouldNotConvertShape:
CouldNotSetShapeIntensity:
CouldNotFlattenShape:
CouldNotAddShape:

    return grErr;
}

```

Printing Extensions

The `AddBackwash` function stores the background picture as a resource in the spool file. `AddBackwash` first calls a local function, `LoadAPict`, to read a picture file and create a `PicHandle` for it. The `LoadAPict` function, which opens the specified file, creates the handle, skips over the file header information, and reads the picture data into the handle. `AddBackwash` then converts the picture into a QuickDraw GX shape by calling the `GXConvertPICTToShape` function, which is described in *Inside Macintosh: QuickDraw GX Objects*.

Once `AddBackwash` has a shape object that represents the background picture, it calls a local function, `SetShapeIntensity`, to lighten the background picture to the intensity level selected by the user (in the background picture panel). `SetShapeIntensity` modifies the transfer object for each shape in the background picture to use blending, which lightens the final picture. The intensity is adjusted to match the value selected by the user.

Finally, `AddBackwash` flattens the picture shape into a handle by calling the local function `ShapeToHandle` and adds the flattened shape to the spool file as a resource. The resource is added in the system heap and is marked as purgeable. `ShapeToHandle` calls the `GXFlattenShape` function, which is described in *Inside Macintosh: QuickDraw GX Objects*.

Adding the Background Picture to a Page

The background picture printing extension draws the background picture on each page as the page is being despoiled. It does this by overriding the `GXDespoolPage` message with the `BWDespoolPage` function, which is shown in Listing 2-16. This function first forwards the `GXDespoolPage` message, which is described on page 4-75 in the chapter “Printing Messages,” so that any other message handlers that are going to modify the page can do so first, which allows the background picture to be drawn using the final shape of the page.

Listing 2-16 The `BWDespoolPage` function

```
OSErr BWDespoolPage(gxSpoolFile theSpoolFile, long thePageNum,
    gxFormat theFormat, gxShape *thePage, Boolean *formatChanged)
{
    OSErr          anOSErr;
    gxGraphicsError grErr;
    Handle          vpListHdl, gxShapeHdl = nil;
    gxRectangle     pBounds;
    Fixed           cx, cy, px, py;
    long            numViewPorts;
    BackwashCollection backwashConfig;

    /*
       First, forward the message to make sure that the final page
```

Printing Extensions

```

    shape is being despoiled.
*/

    anOSError = Forward_GXDespoolPage (theSpoolFile,
                                       thePageNum, theFormat, thePage, formatChanged);
    nrequire((grErr = (gxGraphicsError) anOSError),
            ForwardMessage_Failed);

    grErr = GetJobCollectionItem(&backwashConfig, nil,
                                kBackwashCollectionType, kBackwashSettingsID);
    nrequire_action((!grErr && backwashConfig.addBackwash
                    && backwashConfig.haveFileInfo),
                    NotAddingBackwash, grErr = noErr);
/*
    Load the flattened shape resource (the background picture) if
    necessary. This is only needed for the first page because the
    shape reference is stored in a global variable.
*/
    if (gBackwashShape == nil)
        anOSError = Send_GXDespoolResource(theSpoolFile,
                                           kBackwashCollectionType, r_BackwasPICTID, &gxShapeHdl);
    nrequire((grErr = (gxGraphicsError) anOSError),
            DespoolResource_Failed);
/*
    If necessary, unflatten the background picture shape. Since a
    view port list is needed for the unflattening, use the current
    page's view port list.
*/
    if (gBackwashShape == nil)    /* first page */
    {
        numViewPorts = GXGetShapeViewPorts(*thePage, nil);
        vpListHdl = TempNewHandle(numViewPorts * sizeof(gxViewPort),
                                  &anOSError);

        grErr = (gxGraphicsError) anOSError;
        nrequire_action(grErr, TempNewHandle_Failed,
                        ReleaseResource(gxShapeHdl));
        HLock(vpListHdl);
        GXGetShapeViewPorts(*thePage, (gxViewPort *) *vpListHdl);
        gBackwashShape = HandleToShape(gxShapeHdl, numViewPorts,
                                       (gxViewPort *) *vpListHdl);
        DisposHandle(vpListHdl); /* all done with this */

        if (gxShapeHdl)

```

Printing Extensions

```

        ReleaseResource(gxShapeHdl);
        nrequire(GXGetGraphicsError(&grErr), CouldNotSetUpShape);
    }
/*
    Now use the current page format to obtain the page dimensions.
    Use the dimensions and the bounds of the picture shape to
    center the page. Once the shape is centered, add it behind the
    shapes that constitute the contents of the page.
*/
    GXGetFormatDimensions(theFormat, &pBounds, nil);
    grErr = GXGetJobError(GXGetJob());
    nrequire(grErr, GetFormatDims_Failed);

    cx = pBounds.left + (pBounds.right - pBounds.left) >>1;
    cy = pBounds.top + (pBounds.bottom - pBounds.top) >>1;

    GXGetShapeBounds(gBackwashShape, 0, &pBounds);
    px = pBounds.left + (pBounds.right - pBounds.left) >>1;
    py = pBounds.top + (pBounds.bottom - pBounds.top) >>1;
    GXMoveShapeTo(gBackwashShape, cx-px, cy-py);

    GXSetPictureParts(*thePage, 1, 0, 1, &gBackwashShape,
                                                              nil, nil, nil);

    GXGetGraphicsError(&grErr);

ForwardMessage_Failed:
NotAddingBackwash:
DespoolResource_Failed:
TempNewHandle_Failed:
CouldNotSetUpShape:
GetFormatDims_Failed:

    return grErr;
}

```

After forwarding the `GXDespoolPage` message, `BWDespoolPage` makes sure that the background-picture-configuration collection item is available in the job collection. If the item is not found in the job collection, it means that the background picture panel was never opened because that event triggers the adding of the item to the collection. If the panel was never opened, then the user is printing without dialog boxes, and no background picture is added to the pages. In this case, `BWDespoolPage` does not return an error because the function did not really fail. This is accomplished in the `nrequire_action` call, which changes the error code to `noErr` and branches to the end of the function.

If the background-picture-configuration collection item is found, `BWDespoolPage` checks to see if the global variable for the background picture shape is `nil`. If so, `BWDespoolPage` reads the resource from the spool file and unflattens it into a picture shape.

Finally, `BWDespoolPage` centers the background picture shape on the page and adds the shape to the page contents.

Closing the Spool File

When QuickDraw GX finishes with the spooling of a document, it sends the `GXCloseSpoolFile` message, which is described on page 4-79 in the chapter “Printing Messages.” The background picture printing extension overrides this message with the `BWCloseSpoolFile` function, shown in Listing 2-17, to dispose of the picture shape that it allocated for the background picture.

Listing 2-17 The `BWCloseSpoolFile` override function

```
OSErr BWCloseSpoolFile(gxSpoolFile theSpoolFile,
                      long closeOptions)
{
    if (gBackwashShape != nil)
    {
        GXDisposeShape(gBackwashShape);
        gBackwashShape = nil;
    }
    return Forward_GXCloseSpoolFile(theSpoolFile, closeOptions);
}
```

Shutting Down the Background Picture Extension Environment

When the background picture printing extension is finished, it needs to deallocate the globals that it allocated in its override of the `GXInitialize` message. In the background picture extension, the override of the `GXShutDown` message, `BWShutDown`, calls the `DisposeMessageGlobals` function to do that deallocation, as shown in Listing 2-18. The `GXShutDown` message is described on page 4-44 in the chapter “Printing Messages.”

Listing 2-18 The `BWShutDown` override function

```
OSErr BWShutDown()
{
    DisposeMessageGlobals();
    return noErr;
}
```

Using Resources in Printing Extensions

You must create a number of resources that define your printing extension and provide QuickDraw GX with the information that it needs to properly load and execute the extension. The resources an extension contains are defined in a printing extension file of type 'pext'. Each printing extension file must include certain resources and may include some optional resources, as shown in Table 2-5.

Table 2-5 Resource types used to define a printing extension

Resource type	Count	Description
'vers'	1 or more	Records version and compatibility information for your extension
'scop'	1 or more	Defines with which devices and drivers the extension is compatible
'eopt'	1	Tells QuickDraw GX when to load the extension
'over'	1-4	Defines which messages your extension needs to receive
'load'	1	Specifies the order in which extensions are installed in the message chain
'stat'	0 or more	Defines status messages for display
'plrt'	0 or more	Defines printing alert messages for display
'ppnl'	0 or more	Defines the contents of a panel that you are adding to a dialog box
'xdtl'	0 or more	Provides information that QuickDraw GX needs to execute panel controls
'ptyp'	0 or more	Defines characteristics of a paper type

Most of these resource types are described in the chapter “Printing Resources” in this book. The panel ('ppnl'), extended panel ('xdtl'), and paper type ('ptyp') resources are described in *Inside Macintosh: QuickDraw GX Printing*. This section provides examples of these resources as used in the background picture printing extension. The contents of the file `backwash.r`, which contains the resource definitions for the background picture printing extension, are shown in the QuickDraw GX sample code.

To avoid conflicts with resources defined by printer drivers, application programs, and Macintosh system software, the resources that you define for your printing extension must have IDs in the range 0x9600 (-27136) through 0x97FF (-26625). Most of the resources have specific ID values that you must use with them; these IDs are shown in the examples in this chapter and are defined in the resource descriptions in the chapter “Printing Resources” in this book.

All of the resources that you define for your printing extensions need to be loaded into the system heap and need to be purgeable. System resources are stored in the system heap as opposed to the application heap, where application resources are stored. Purgeable resources can be purged by the Memory Manager when space is required, as described in *Inside Macintosh: Memory*. You need to specify these attributes in the first line of every resource that you define for your extensions, as is done in every resource example in this chapter.

Defining Code Segments in Your Printing Extension

The code segments that you use to create your printing extension must use segment type 'pext', which is defined by the constant `gxPrintingExtensionType`. The ID of your first code segment needs to be 0, and you need to increment the ID by 1 for each subsequent segment in your extension.

Defining Version Compatibility for Your Printing Extension

Your printing extension must contain at least one version ('vers') resource that defines its compatibility with QuickDraw GX. Version resources are used to record version information for Macintosh applications. You need to include a version resource with an ID of `gxPrintingExtensionBaseID` that defines with which version of QuickDraw GX your extension is compatible.

IMPORTANT

You must include the version resource or your extension will not load. ▲

For the current version, the value of the first byte of the resource definition must be either 1 or 0. Listing 2-19 shows the version resource that defines QuickDraw GX compatibility for the background picture printing extension.

Listing 2-19 The QuickDraw GX version resource for the background picture printing extension

```
resource 'vers' (gxPrintingExtensionBaseID, sysHeap, purgeable) {
    0x0,
    0x0,
    release,
    0x0,
    verUS,
    " ",
    " "
};
```

You can also include standard version resources in the resource files for your printing extension. These resources are described in *Inside Macintosh: Macintosh Toolbox Essentials*. Listing 2-20 shows the standard version resources for the background picture printing extension.

Listing 2-20 The standard version resources for the background picture printing extension

```
resource 'vers' (1, sysHeap, purgeable) {
    0x1,
    0x0,
    beta,
    0x2,
    verUS,
    "1.0b2",
    "1.0b2, © Apple Computer, Inc. 1992-1993"
};

resource 'vers' (2, sysHeap, purgeable) {
    0x1,
    0x0,
    beta,
    0x2,
    verUS,
    "1.0b2",
    "Backwash v1.0b2"
};
```

Defining the Scope of Your Printing Extension

When you write a printing extension, you must define the types of devices and drivers with which your extension is compatible. You do this by including an extension scope ('`scop`') resource in your extension. The extension scope resource is described in detail on page 6-26 in the chapter "Printing Resources."

The scope of your printing extension can range from very general to quite specific. The scope of some extensions is known as universal scope, which means that the extension can run on any device. The scope of other extensions is defined in terms of the specific devices on which the extension can run. And the scope of yet other extensions is defined in terms of the specific devices on which the extension cannot run. Many extensions that affect only the final image on the printed page can be defined with universal scope.

There are three kinds of extension scope resources used in printing extensions, each of which has a unique ID defined for it. The constants for these IDs are shown in Table 2-6.

Table 2-6 Identifiers for the extension scope resource

Constant	Explanation
gxDriverScopeID	Defines with which imaging systems the extension is compatible. Four imaging systems types are supported: raster, PostScript, vector, and universal.
gxPrinterScopeID	Defines specific devices with which the extension is compatible.
gxPrinterExceptionScopeID	Defines specific devices with which the extension is not compatible.

The declarations of these and other constants that are used in printing extension resources are shown in the section “Summary of Printing Resources” beginning on page 6-90 in the chapter “Printing Resources.”

The background picture printing extension is compatible with all raster and PostScript printing devices. Its extension scope resource is defined as shown in Listing 2-21.

Listing 2-21 The extension scope resource for the background picture printing extension

```
resource gxExtensionScopeType (gxDriverScopeID,sysHeap,
                                purgeable)
{
    {
        'post',          /* compatible with PostScript printers */
        'rast'          /* compatible with raster printers */
    };
};
```

You can include multiple extension scope resources in your printing extension file to pinpoint the devices on which the extension runs. Typically, you use one extension scope resource to define the imaging systems that your extension is compatible with, and you add other extension scope resources to include or exclude specific devices. Listing 2-22 includes three extension scope resources.

Listing 2-22 An example of extension scope resources

```

resource gxExtensionScopeType (gxDriverScopeID, sysHeap,
                                purgeable)
{
    {
        'vect';          /* compatible with all vector devices */
        'post';          /* compatible with all PostScript devices */
    };
};

resource gxExtensionScopeType (gxPrinterScopeID, sysHeap,
                                purgeable)
{
    {
        'lwsc';          /* also compatible w/Personal LaserWriter SC */
    };
};

resource gxExtensionScopeType (gxPrinterExceptionScopeID, sysHeap,
                                purgeable)
{
    {
        'odd1';          /* not compatible with odd1 device */
    };
};

```

The resources in Listing 2-22 indicate that the printing extension is compatible with all PostScript and vector devices except for the one that is identified as 'odd1' and with no raster devices except for the Personal LaserWriter SC.

Optimizing the Use of Your Extension

You must include an extension optimization ('eopt') resource in your printing extension file to provide QuickDraw GX with information about when the extension needs to be loaded into memory. This information makes it possible to optimize the use of printing extensions, thus maximizing performance during the printing process. The optimization resource has an ID of `gxExtensionOptimizationID`.

QuickDraw GX also uses the values in this resource to determine if your extension has to be available on the user's system or on a print server if the user is printing from a print server on a network. For example, an extension that overrides messages during the despooling process needs to be available on the system that is communicating with the printer, while an extension that overrides messages during the spooling process needs to be available on the system that is creating the spool file. You need to carefully choose which of the optimization flags to include in the extension optimization resource for

your printing extension. Including the wrong flag can cause performance degradation or worse, so choose carefully from among the flag values. The constants for these flags, each of which is a Boolean value, are shown in Table 6-11 on page 6-30 in the chapter “Printing Resources.”

Listing 2-23 shows the extension optimization resource for the background picture printing extension, which modifies each page of a document as it is despoiled.

Listing 2-23 An example of an extension optimization resource

```
resource 'eopt' (gxExtensionOptimizationID, sysHeap, purgeable)
{
    {
        gxDontExecuteDuringImaging,
        gxDontNeedDeviceStatus,
        gxChangePageAtGXDespoolPage,
        gxDontChangePageAtGXImagePage,
        gxDontChangePageAtGXRenderPage,
        gxNotServerPresenceRequired,
        gxClientPresenceRequired
    };
};
```

The despooling process during the imaging phase of printing is described in the chapter “Introduction to Printing Extensions and Drivers” in this book. The `GXDespoolPage` message is described on page 4-75, the `GXImagePage` message is described on page 4-94, and the `GXRenderPage` message is described on page 4-96 in the chapter “Printing Messages.”

Specifying Which Messages Your Extension Overrides

You must include an override (`'over'`) resource in your printing extension file to provide QuickDraw GX with a list of the messages that your printing extension is overriding. Each entry in the override resource specifies a message and information about the resource in which the code segment for the message override is found. You typically include separate override resources for universal messages and for messages specific to an imaging system, as shown in Listing 2-24.

The code segment information for each message includes the resource ID of the code segment and the offset into the code segment for the instruction to jump to the message override function. The first 4 bytes of the code segment are reserved for use by QuickDraw GX and must be 0; thus, the first offset location is 4.

Each override resource in a printing extension has an ID of 0 for messages specific to an imaging system and an ID of `gxExtensionOverrideID` (-1) for universal messages. Unlike printer drivers, printing extensions may not override messages that QuickDraw GX sends to ensure compatibility with the Macintosh Printing Manager.

Listing 2-24 Override resources for a printing extension

```

#define segmentID NewSegmentID    /* defined as part of the make */
#define firstOffset 4
#define kRasterMsgsOVERRIDEID gxPrintingDriverBaseID+1

resource gxOverrideType (gxExtensionUniversalOverrideID, sysHeap,
                        purgeable)
{
    {
        gxInitialize,      segmentID, firstOffset+0,
        gxShutDown,        segmentID, firstOffset+4,
        gxDespoolPage,     segmentID, firstOffset+8,
        gxSetupImageData,  segmentID, firstOffset+12,
        gxOpenConnection,  segmentID, firstOffset+16,
        gxCloseConnection, segmentID, firstOffset+20,
        gxStartSendPage,   segmentID, firstOffset+24,
        gxFinishSendPage,  segmentID, firstOffset+28,
        gxDefaultPrinter,  segmentID, firstOffset+32,
        gxWriteData,       segmentID, firstOffset+36,
        gxCheckStatus,     segmentID, firstOffset+40,
        gxGetDeviceStatus, segmentID, firstOffset+44,
        gxCreateImageFile, segmentID, firstOffset+48,
        gxFetchTaggedData, segmentID, firstOffset+52,
    };
};

resource gxOverrideType (gxExtensionImagingOverrideID, sysHeap,
                        purgeable)
{
    {
        gxRasterDataIn,    segmentID, firstOffset+56,
        gxRasterLineFeed,  segmentID, firstOffset+60,
    };
};

```

The first resource in Listing 2-24 lists the universal messages that the extension overrides, and the second resource specifies the messages that are specific to a raster imaging system and that the extension overrides. Each message listed in the resources specifies the name of the message, the ID of that segment, and where to find the message in the jump table. The ID of the code segment for all of these messages is defined by the constant `segmentID`. The jump statements to the code that implements the overrides are each 4 bytes long. The first is found at the first offset location (byte 4), and each subsequent jump statement is found 4 bytes beyond the previous one.

Defining the Loading Order of Your Extension

You also need to include an extension load ('load') resource for your printing extension to tell QuickDraw GX the default value that specifies where to load your extension into the printing message chain. If you have one or more message overrides that prefer to handle a message before or after other handlers, you can specify that in this resource. The value that you define in your extension load resource specifies the default loading order for your extension when it is first added to the system. The user can modify the loading order of extensions by using the Extension Setup dialog box.

Note

The value that you specify in the extension load resource for your extension provides a default. It is used by QuickDraw GX as a hint. Because the user can change the order, you do not have any guarantee regarding the load order of your extension. ♦

The constants for the values that you can specify in your extension load resource are shown in Table 2-7.

Table 2-7 Loading order constants for an extension

Constant	Explanation
<code>gxExtensionLoadFirst</code>	This extension is loaded as the top message handler in the printing message chain
<code>gxExtensionLoadAnywhere</code>	This extension can be loaded anywhere in the print message chain because its message overrides don't depend on a handling order
<code>gxExtensionLoadLast</code>	This extension is loaded as the bottom message handler in the print message chain

The extension load resource for the background picture printing extension is shown in Listing 2-25.

Listing 2-25 The extension load resource for the background picture extension

```
resource gxExtensionLoadType (gxExtensionLoadID, sysHeap, purgeable)
{
    gxExtensionLoadFirst
};
```

The background picture extension draws the background picture on each page after the page has been fully composed. Thus, this printing extension should be loaded first in the message chain, which means that it receives messages after all other message handlers. Then the printer driver and any other message handler can modify the page contents at despool time prior to the background picture being drawn.

