CHAPTER 9

Printing Manager

Contents

This chapter describes how your application can use the Printing Manager to perform QuickDraw-based printing on a printer connected to a Macintosh computer. The Printing Manager works with the printer driver for the currently selected printer so that your application can draw an image on a printer just as it draws an image on a screen. This allows your application to use the same QuickDraw routines for printing as for screen display.

You should read this chapter if your application allows the user to print. If you want to print with features—such as rotated text and hairlines—that are not supported by QuickDraw, you should read Appendix B, "Using Picture Comments for Printing."

Before reading this chapter, you should be familiar with QuickDraw's drawing routines and the `GrafPort` and `CGrafPort` data types, as described in the chapters "Basic QuickDraw," "QuickDraw Drawing," and "Color QuickDraw" in this book. You may also need to refer to *Inside Macintosh: Text* for information about printing text from non-Roman script systems.

About the Printing Manager

The **Printing Manager** is a collection of system software routines that your application can use to print from the Macintosh computer to any type of connected printer. The Printing Manager is available on all Macintosh computers. When printing, your application calls the same Printing Manager routines regardless of the type of printer selected by the user.

When you print a document using the Printing Manager, the Printing Manager uses a printer driver to do the actual printing. A **printer driver** does any necessary translation of QuickDraw drawing routines and—when requested by your application—sends the translated instructions and data to the printer. Printer drivers are stored in **printer resource files,** which are located in the Extensions folder inside the System Folder. Each type of printer has its own printer driver. One printer driver can communicate with several printers of the same type; for example, the LaserWriter printer driver can work with multiple LaserWriter printers on a network.

The **current printer** is the printer that the user last selected from the Chooser. It is the printer driver for the current printer that actually implements the routines defined by the Printing Manager. Every Printing Manager routine you call determines the current printer from a resource in the System file and then dispatches your call to the printer driver for that printer.

To print a document, your application uses the `PrOpen` procedure to open the driver for the current printer. Your application then uses the `PrOpenDoc` function to open a **printing graphics port,** which is a data structure of type `TPrPort`; it consists of a QuickDraw graphics port (either a `GrafPort` or `CGrafPort` record) plus additional information. For each page in a document, your application uses the `PrOpenPage` procedure to open the page. Your application then uses QuickDraw routines to draw onto the page.

Ideally, your application should be device-independent, so that when it prints a document, it doesn't rely on the presence of any one printer feature. In general, there are two types of printer drivers: those for QuickDraw printers and those for PostScript printers. QuickDraw printer drivers render images using QuickDraw and then send the rendered images to the printer as bitmaps or pixel maps. PostScript printer drivers convert QuickDraw operations into equivalent PostScript operations, as necessary. The driver sends the converted PostScript drawing operations to the printer, which renders the images by interpreting these operations.

For most applications, sending QuickDraw's picture-drawing routines to the printer driver is sufficient: the driver either uses QuickDraw or converts the drawing routines to PostScript. For some applications, such as page-layout programs, this may not be sufficient; such applications may rely on printer drivers to provide several features that are not available, or are difficult to achieve, using QuickDraw.

If your application requires these features (such as rotated text and dashed lines), you may want to create two versions of your drawing code: one that uses picture comments to take advantage of these features on capable printers, and another that provides QuickDraw-based approximations of these features. Created with the QuickDraw procedure `PicComment`, **picture comments** are data or commands used for special processing by output devices, such as printer drivers. Picture comments may be included in the code an application sends to a printer driver, or stored in the definition of a picture. For more information, see Appendix B, "Using Picture Comments for Printing," in this book.

For information about how the PostScript language works and the specifics of PostScript commands, see the *PostScript Language Reference Manual*, second edition, published by Addison-Wesley.

The Printing Graphics Port

You use the `PrOpenDoc` function to open a document for printing. The `PrOpenDoc` function in turn opens a printing graphics port and returns a pointer to a `TPrPort` record, which defines the printing graphics port.

```
TYPE
   TPPrPort =  ^TPrPort;
   TPrPort  =  {printing graphics port record}
   RECORD
      gPort:      GrafPort;   {graphics port for printing}
      gProcs:     QDProcs;    {procedures for printing in the }
                              { graphics port}
      {more fields for internal use}
   END;
```

The graphics port in the `gPort` field is either a `CGrafPort` or `GrafPort` record, depending on whether the current printer supports color and grayscale and whether Color QuickDraw is available on the computer. If you need to determine the type of graphics port, you can check the high bit in the `rowBytes` field of the record contained in the `gPort` field; if this bit is set, the printing graphics port is based on a `CGrafPort` record.

You print text and graphics by drawing into a printing graphics port using QuickDraw drawing routines, just as if you were drawing on the screen. The printer driver installs its own versions of QuickDraw's low-level drawing routines in the `gProcs` field of the `TPrPort` record. Your calls to high-level QuickDraw routines then drive the printer instead of drawing on the screen.

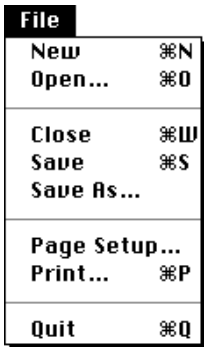
As you draw each page of a document into the printing graphics port, the printer driver translates the calls to QuickDraw routines into the equivalent instructions for the printer. The printer itself does nothing except draw the document on a page, exactly as the printer driver directs it.

Before ever printing a document, however, your application must obtain various printing preferences from the user—usually when the user chooses the Page Setup or Print command from the File menu.

Getting Printing Preferences From the User

If it's likely that a user will want to print the data created with your application, you should support the Page Setup command and the Print command in the File menu. Figure 9-1 shows a typical File menu that includes the Page Setup and Print commands. (See the chapter "Menu Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for detailed information about setting up a File menu with these commands.)

Figure 9-1 A standard File menu for an application



In response to the Page Setup command, your application should display the current printer's **style dialog box,** which allows the user to specify the printing options—such as the paper size and the printing orientation—that your application needs for formatting the document in the frontmost window. In response to the Print command, your application should display the current printer's **job dialog box,** which solicits from the user printing information—such as the number of copies to print, the print quality, and the range of pages to print—for the document in the frontmost window. Each printer driver defines its own style dialog box and job dialog box. Your application can also provide other printing options in these dialog boxes when appropriate.

A `TPrint` record contains the information about the user's choices made with the style and job dialog boxes. When the user saves a document, your application should save the `TPrint` record associated with that document. This allows your application to resume using any style preferences that the user has selected for printing that document. While only the information the user specifies through the style dialog box should be preserved each time the user prints the document, you can save the entire `TPrint` record when you save the document. The information supplied by the user in the job dialog box should pertain to the document only while the document prints; you should not reuse this information if the user prints the document again.

The values that the user specifies through the style dialog box apply only to the printing of the document in the active—that is, frontmost—window. In general, the user should have to specify these preferences only once per document, although the user can choose to change these settings at any time. Figure 9-2 shows the StyleWriter printer driver's style dialog box, displayed by an application in response to the Page Setup command.

Figure 9-2 The style dialog box for a StyleWriter printer

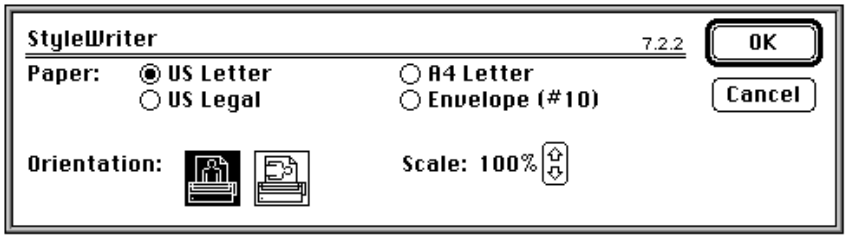
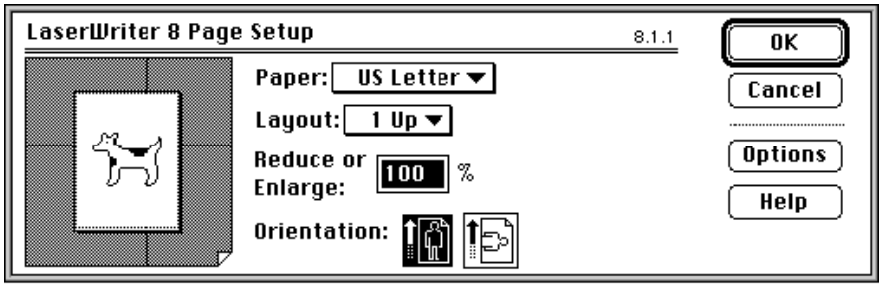


Figure 9-3 shows the style dialog box for a LaserWriter printer. Because each printer resource file defines its own style dialog box, a style dialog box for one printer may differ slightly from that of another printer (as you can see by comparing Figure 9-2 with Figure 9-3).

Figure 9-3 The style dialog box for a LaserWriter printer



You use the `PrStlDialog` function to display the style dialog box defined by the resource file of the current printer. The `PrStlDialog` function handles all user interaction in the items defined by the printer driver until the user clicks the OK or Cancel button. You must call the `PrOpen` procedure prior to calling `PrStlDialog` because the current printer driver must be open for your application to successfully call `PrStlDialog`.

Figure 9-4 shows an example of a job dialog box. Your application should print the document in the active window if the user clicks the Print button.

Figure 9-4 The job dialog box for a StyleWriter printer

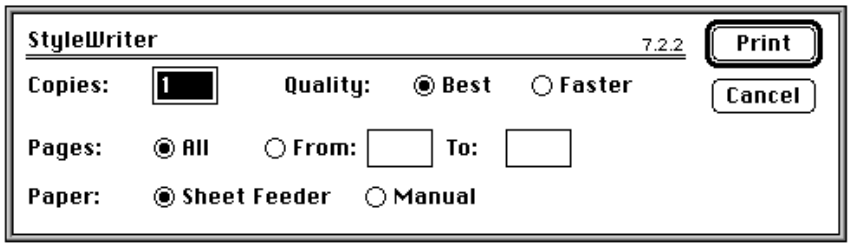
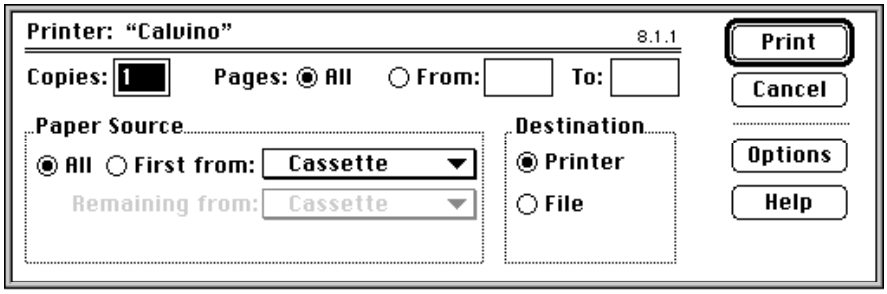


Figure 9-5 shows a job dialog box for a LaserWriter printer.

Figure 9-5 The job dialog box for a LaserWriter printer



Your application uses the `PrJobDialog` function to display the job dialog box defined by the resource file of the current printer. The `PrJobDialog` function handles all user interaction in the items defined by the printer driver until the user clicks the Print or Cancel button.

Your application can customize the style dialog box and the job dialog box to ask for additional information. (Figure 9-12 on page 9-35 shows a print job dialog box that includes two extra checkboxes.) If you customize the style or job dialog box, you must provide a function that handles events, such as clicks, in any items that you add to the dialog box, and you should provide an event filter function to handle events not handled by the Dialog Manager in a modal dialog box.

QuickDraw and PostScript Printer Drivers

There are two main types of printer drivers for Macintosh computers: QuickDraw printer drivers and PostScript printer drivers.

Using QuickDraw drawing operations, **QuickDraw printer drivers** render images on the Macintosh computer and then send the rendered images to the printer in the form of bitmaps or pixel maps. The printer might be a dot-matrix printer, an ink jet printer, a laser printer, or a plotter.

QuickDraw printers are not required to have any intelligent rendering capabilities. Instead, they simply accept instructions from printer drivers to place dots on the page in specific places. A QuickDraw printer driver captures in a temporary disk file (called a **spool file**)—or in memory—the images of an entire page, translates the pixels into dot-placement instructions, and sends these instructions to the printer.

Note
The internal format of spool files is private to the printer drivers and may vary from one printer driver to another. You should not attempt to determine or manipulate the format of these files. ◆

QuickDraw printers are relatively inexpensive to produce because they don't require sophisticated rendering capabilities—instead, they rely on the rendering capabilities of the Macintosh computer. However, QuickDraw printers are also relatively slow. Over 7 million pixels are required to render an 8-by-10-inch image at 300 dpi. Many QuickDraw printers use some form of data compression to improve their performance. For the ImageWriter printer, for example, the printer driver instructs the printer only where to place ink; the printer driver assumes that the rest of the page should remain untouched. Nevertheless, nearly 900 KB is required to render a full-page image at 1 bit per pixel. A color printer that uses 8 bits per pixel requires eight times as much data. Such large memory requirements may require the driver to process the image in horizontal strips, or bands, which further impairs printing speed.

PostScript printer drivers, on the other hand, convert QuickDraw drawing operations into equivalent PostScript drawing operations, as necessary. PostScript printers have their own rendering capabilities. Instead of rendering an entire page on the Macintosh computer and sending all the pixels to the printer, PostScript printer drivers typically send drawing commands directly to the printer, which itself renders images on the page. Many of Apple's LaserWriter printers use the PostScript page-description language to render images in this way, thereby off-loading image processing from the computer. This gives PostScript printers a speed advantage over QuickDraw printers.

QuickDraw printer drivers must capture an entire page before sending any of it to the printer, but PostScript printer drivers may be able to send commands as soon as they are generated to printers. This can result in faster printing, but it doesn't allow the printer driver to examine entire pages for their use of color, fonts, or other resources that the printer needs to have specially processed. Therefore, some PostScript printer drivers may capture page images in a spool file so that the driver can analyze the pages before sending them to the printer.

Some printer drivers allow users to specify **background printing,** which allows the user to work with an application while documents are printing. These printer drivers, which can be either QuickDraw or PostScript, send printing data to a spool file in the PrintMonitor Documents folder inside the System Folder. Do not confuse the different uses of these various spool files. With background printing, print files are spooled to disk to allow the user to work with an application while documents are printing; many printer drivers support background printing regardless of their other capabilities. Some printer drivers create spool files while processing a page image—this, however, does not allow the user to work with the application while the document is printing.

IMPORTANT

There is no reliable manner in which you can determine whether a printer driver creates a spool file—whether for processing of a page image or for background printing. With the exception of using the `PrPicFile` procedure, described on page 9-71, your print loop should not base any of its actions on whether a printer driver creates a spool file. ▲

Page and Paper Rectangles

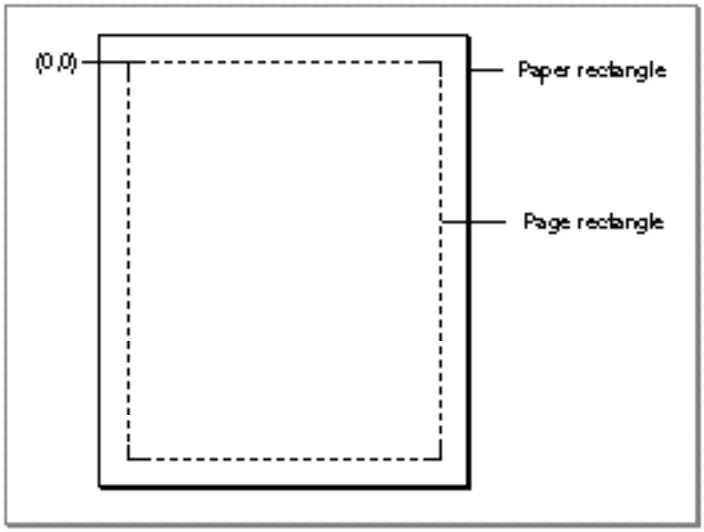
When printing a document, you should consider these two aspects of the layout of the page:

- the physical size of the paper
- the area on the paper that the printer can use to format the document, which is usually smaller than the physical sheet of paper to account for margins or the mechanical limitations of the printer

The **page rectangle** represents the boundaries of the printable area on a page. Its upper-left corner always has the coordinates (0,0). The coordinates of the lower-right corner give the maximum page height and width attainable on the given printer; these coordinates are specified by the units used to express the resolution of the printing graphics port. For example, the lower-right corner of a page rectangle used by the PostScript LaserWriter printer driver for an 8.5-by-11-inch U.S. letter page is (730,552) at 72 dpi.

The **paper rectangle** gives the physical paper size, defined in the same coordinate system as the page rectangle. Thus, the upper-left coordinates of the paper rectangle are typically negative, and its lower-right coordinates are greater than those of the page rectangle. Figure 9-6 shows the relationship of these two rectangles.

Figure 9-6 Page and paper rectangles



Your application should always use the page rectangle sizes provided by the printer driver and should not attempt to change them or add new ones. If your application offers page sizes other than those provided by the printer driver for the current printer, you risk compatibility problems.

When formatting a page for printing, remember to use the page rectangle size that the user has chosen to format the document. (See "The TPrint Record and the Printing Loop" on page 9-11 for more information about where to find the user's choices for formatting the document.)

Printer Resolution

Resolution refers to the degree of detail at which a device such as a printer or a screen can display an image. Resolution is usually specified in dots per inch, or *dpi*, in the x and y directions. The higher the value, the finer the detail of the image.

A printer driver supports either discrete or variable resolution. If a printer driver supports **discrete resolution,** an application can choose from only a limited number of resolutions that are predefined by the printer driver. For example, the ImageWriter printer driver supports four discrete resolutions: 72 by 72 dpi, 144 by 144 dpi, 80 by 72 dpi, and 160 by 144 dpi.

If a printer driver supports **variable resolution,** an application can define any resolution within a range bounded by maximum and minimum values defined by the printer driver. LaserWriter printer drivers support variable resolution within a range from 25 dpi to 1500 dpi in both the x and y directions.

To print, your application does not need to check the resolutions available or set the resolution. However, if your application does factor in possible resolutions, it can obtain the best quality output from a printer by choosing equal resolutions for the x and y directions. For information on how to determine the available resolution or resolutions for the currently selected printer, see "Determining and Setting the Resolution of the Current Printer" on page 9-30.

The TPrint Record and the Printing Loop

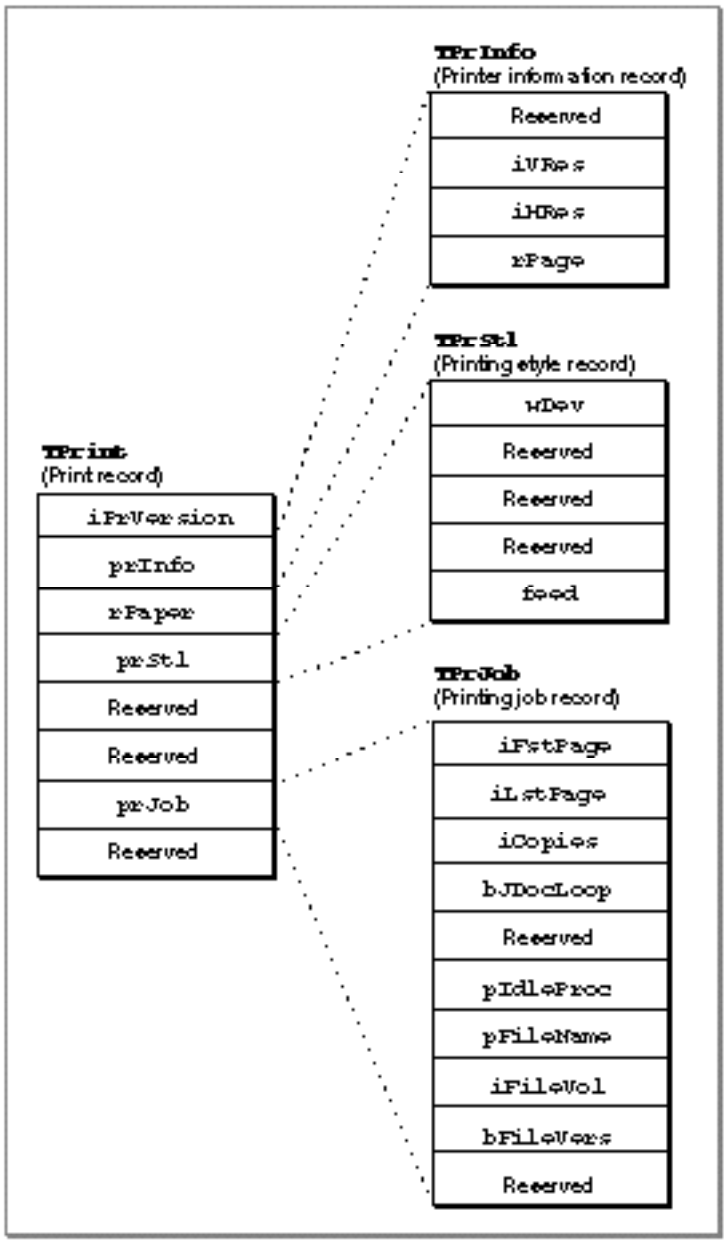
To print a document, you need to create a print record. The **`TPrint` record** is a data structure of type `TPrint`, and it contains fields that specify the Printing Manager version, information about the printer (such as its resolution in dpi), and the dimensions of the paper rectangle. Most Printing Manager routines require that you provide a handle to a `TPrint` record as a parameter.

Your application allocates the memory for a `TPrint` record (using the Memory Manager function `NewHandle`, for example) and then initializes the new `TPrint` record using the `PrintDefault` procedure. (Your application can also validate an existing `TPrint` record by using the `PrValidate` function.) When the user chooses the Print command, your application passes a handle to a `TPrint` record to the `PrJobDialog` or `PrDlgMain` function to display a job dialog box to the user; the function alters the `TPrint` record according to the user's responses.

When the user chooses the Page Setup command, your application passes a handle to a `TPrint` record to the `PrStlDialog` or `PrDlgMain` function to display a style dialog box to the user; the function alters the `TPrint` record according to the user's responses.

The `TPrint` record contains several other records, as illustrated in Figure 9-7. The `TPrInfo` record, which is a data structure of type `TPrInfo`, gives you the information needed for page composition, including the vertical and horizontal resolutions of the printer in dpi and the boundaries of the page rectangle. The **`TPrJob` record,** which is a data structure of type `TPrJob`, gives you information about a particular print job—for instance, the first and last pages to be printed, the number of copies, and the method of printing that the Printing Manager will use.

Figure 9-7 A `TPrint` record



The `PrJobDialog`, `PrStlDialog`, and `PrDlgMain` functions alter the appropriate fields of the `TPrint` record. In particular, the `PrJobDialog` function alters the `prJob` field (which contains a `TPrJob` record), and the `PrStlDialog` function alters the `prInfo` field (which contains a `TPrInfo` record). The `PrDlgMain` function alters either field, depending on whether you use the function to display a job or a style dialog box.

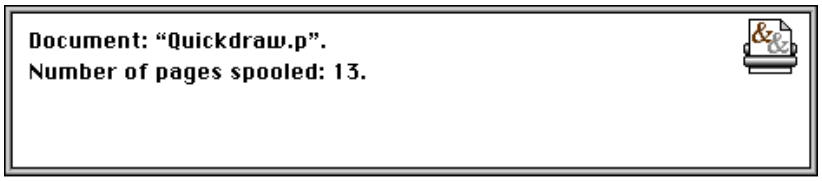
▲ **WARNING**
Your application should not directly change the user-supplied data in the `TPrint` record; it should use the `PrStlDialog` and `PrJobDialog` functions or the `PrDlgMain` function to allow the user to specify printing options that the printer driver then translates to the appropriate fields in the `TPrint` record. The only fields you may need to set directly are those containing optional information in the `TPrJob` record (for example, the `pIdleProc` field, which contains a pointer to an idle procedure). Attempting to set other values directly in the `TPrint` record can produce unexpected results. ▲

Your program code should contain a **printing loop** that handles your printing needs, including presenting the job dialog box and determining the range of pages to be printed. An example of a printing loop is shown in Listing 9-2 on page 9-20; the structure of a `TPrint` record is defined in detail on page 9-44.

Print Status Dialog Boxes

Because the user must wait for a document to print (that is, the application must draw the data in the printing graphics port and the data must be sent either to the printer or to a spool file before the user can continue working), many printer drivers display a print status dialog box informing the user that the printing process is under way. As shown in Figure 9-8, the print status dialog box usually provides information about the document being printed and indicates the current status of the printing operation.

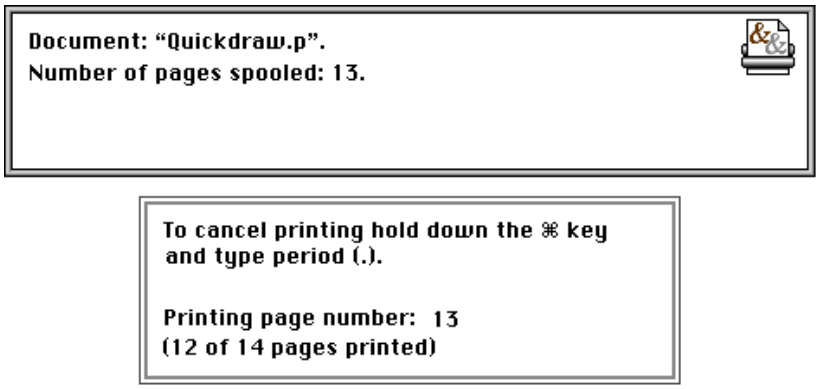
Figure 9-8 The print status dialog box for a LaserWriter printer driver printing in the background



A user should always be able to cancel printing by pressing Command-period. To determine whether the user has canceled printing, the printer driver runs an **idle procedure** whenever it directs output to the printer or to a spool file. The idle procedure takes no parameters and returns no result; the printer driver runs it periodically.

Not all printer drivers, however, display print status dialog boxes. As you can see in Figure 9-8, the print status dialog box for the LaserWriter printer driver (as well as status dialog boxes for many other printer drivers) doesn't inform the user how to cancel the printing operation. Your application can display its own status dialog box that informs the user about the status of the printing operation and how to cancel printing. If the printer driver displays its own print status dialog box, your application's print status dialog box may appear in an inactive window. Figure 9-9 shows an example of an application's print status dialog box that appears in addition to the print status dialog box displayed by the LaserWriter printer driver.

Figure 9-9 A status dialog box with the LaserWriter printer driver's print status dialog box



Note
Your application cannot prevent a printer driver from displaying its own status dialog box, and your application cannot determine where on the screen a printer driver will display its status dialog box. ◆

If your application uses its own print status dialog box, your application should display it just before printing. Your print status dialog box should indicate that the user can press Command-period to cancel printing; your status dialog box may also provide a button that lets the user cancel printing. Your status dialog box should also give information about the document being printed and indicate the current status of the printing operation.

The `TPrJob` record contained in the `TPrint` record contains a pointer to an idle procedure in the `pIdleProc` field. If this field contains the value `NIL`, then the printer driver uses its default idle procedure. The default idle procedure checks for Command-period keyboard events and sets the `iPrAbort` error code if one occurs, so that your application can cancel the print job at the user's request. However, the default idle procedure does not display any dialog boxes. It is up to the printer driver or your application to display a print status dialog box.

To handle update information in your status dialog box during the printing operation, you should install your own idle procedure in the `pIdleProc` field. Your idle procedure should also check whether the user has pressed Command-period, in which case your application should stop its printing operation. If your status dialog box contains a button to cancel the printing operation, your idle procedure should also check for clicks in the button and respond accordingly.

IMPORTANT

In your status dialog box, do not include an option to pause the printing process. Pausing may cause timeout problems when printing to a printer on a network. Communication between the Macintosh computer and the printer must be maintained to prevent a job or a wait timeout. If there is no communication for a period of time (over two minutes, for example, for the PostScript LaserWriter printer), the printer times out and the print job terminates. Because there is no reliable method for determining the type of printer, you should be aware of the possibility of a printer on a network timing out for a user who wants to pause printing for over two minutes. ▲

If you do not supply your own idle procedure, you can determine whether the user has canceled printing by calling the `PrError` function after each call to a Printing Manager routine. The `PrError` function returns the result code `iPrAbort` when the user cancels printing.

See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about creating and displaying dialog boxes. For information about creating an idle procedure, see "Writing an Idle Procedure" on page 9-38.

Using the Printing Manager

The Printing Manager defines routines that give your application device-independent control over the printing process. You can use these routines to print documents, to display and alter the print dialog boxes, and to handle printing errors.

To use the Printing Manager, you must first initialize QuickDraw, the Font Manager, the Window Manager, the Menu Manager, TextEdit, and the Dialog Manager. The first Printing Manager routine to call, when you are ready to print, is `PrOpen`; the last routine to call is `PrClose`.

All of the Printing Manager routines described in this chapter are available on both basic QuickDraw and Color QuickDraw systems using system software version 4.1 or later. However, not all printer drivers support all features provided by the `PrGeneral` procedure. (When you call the `PrGeneral` procedure, described in "Getting and Setting Printer Information" beginning on page 9-28, it in turn calls the current printer driver to get or set the desired information.) After calling `PrGeneral` and passing it a particular opcode, you should call the `PrError` function and test whether it returns the `opNotImpl` result code, which indicates that the printer driver does not support that particular opcode.

All printable documents must have a `TPrint` record. Each `TPrint` record contains information about page size, number of copies requested, and the range of pages the user wants printed. Although only the information the user specifies through the style dialog box should be preserved each time the user prints the document, you can save the entire `TPrint` record when you save the document. The next time the user opens the document, you can retrieve the user's preferences as saved in the `TPrint` record and then use the `PrValidate` function to validate the fields of the `TPrint` record.

To print a user's document, you must first create or validate a `TPrint` record for the document. You can use the `PrintDefault` procedure to initialize the values in a `TPrint` record. You can use the `PrValidate` function to check that an existing `TPrint` record is compatible with the current printer and its driver. Your application should include a printing loop that handles printing and checks for printing errors at every step.

You should never assume the type of printer that has been selected; your application should always be able to print to any type of printer. However, for some special features that are not supported by QuickDraw (notably rotated text and graphics, dashed lines, and hairlines), you may want to create two versions of your drawing code: one that uses picture comments to take advantage of the features, and another that provides QuickDraw-based implementations of these features. Using picture comments, your application can instruct printer drivers to perform operations that QuickDraw does not support. For more information, see Appendix B, "Using Picture Comments for Printing," in this book.

The rest of this section describes how you can

- create and use a `TPrint` record
- structure your printing loop to print a document
- use the `PrGeneral` procedure to determine printer characteristics
- alter the style and job dialog boxes
- write an idle procedure that runs during printing
- handle printing errors

Be aware that the burden of maintaining backward compatibility with early Apple printer models—as well as maintaining compatibility with over a hundred existing printer drivers—requires extra care on your part. When the Printing Manager was initially designed, it was intended to support ImageWriter printers directly attached to Macintosh computers with only a single floppy disk and 128 KB of RAM. Later, the Printing Manager was implemented on PostScript LaserWriter printer drivers for more powerful Macintosh computers sharing LaserWriter printers on networks. Since then, the Printing Manager has been implemented on a substantial—and unanticipated—number of additional Apple and third-party printer drivers, each in its own, slightly unique way. When you use Printing Manager routines and data structures, you should be especially wary of and defensive about possible error conditions. Because Apple has little control over the manner in which third parties support the Printing Manager in their printer drivers, you should test your application's printing code on as many printers as possible.

Creating and Using a TPrint Record

To print a document, you need a valid `TPrint` record that is formatted for the current versions of the Printing Manager and the printer driver.

To create a new `TPrint` record, you must first create a handle to it with a Memory Manager function such as `NewHandle` or `NewHandleClear`. You then must use the `PrintDefault` procedure to set the fields of the record to the default values for the current printer driver, as illustrated in the following code fragment.

```
VAR
   prRecHdl:   THPrint;

   {allocate handle to a TPrint record}
   prRecHdl := THPrint(NewHandleClear(SizeOf(TPrint)));
   IF prRecHdl <> NIL THEN
      PrintDefault(prRecHdl) {sets appropriate default values }
                             { for current printer driver}
   ELSE
   ; {handle error here}
```

You can also use an existing `TPrint` record (for instance, one saved with a document). If you use an existing `TPrint` record, be sure to call the `PrValidate` function before using the `TPrint` record to make sure it's valid for the current version of the Printing Manager and for the current printer driver.

Listing 9-1 shows an application-defined routine that reads a `TPrint` record that the application has saved as a resource of type `'SPRC'` with the document. (The Resource Manager routines `CurResFile`, `UseResFile`, `Get1Resource`, and `DetachResource` that are shown in this listing are described in the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.)

Listing 9-1 Reading a document's `TPrint` record

```
FUNCTION MyGetPrintRecordForThisDoc (refNum: Integer;
                                     VAR prRecHdl: THPrint;
                                     VAR prRecChanged: Boolean):
                                     OSErr;
VAR
   saveResFile:   Integer;
BEGIN
   saveResFile := CurResFile; {save the resource file for the document}
   UseResFile(refNum);
   prRecHdl := THPrint(Get1Resource('SPRC', kDocPrintRec));
   IF prRecHdl <> NIL THEN
```

```
   BEGIN
      DetachResource(Handle(prRecHdl));
      prRecChanged := PrValidate(prRecHdl);  {validate TPrint record}
      MyGetPrintRecordForThisDoc := PrError;
   END
   ELSE
      MyGetPrintRecordForThisDoc := kNILHandlePrintErr;
   UseResFile(saveResFile);
END;
```

You should save the `TPrint` record when the user saves the document. By doing this, you can save any preferences that the user has selected for printing that document, such as orientation of the page or page size. See the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox* for information about saving data such as `TPrint` records in resources.

Every printer driver uses the fields of the `TPrint` record differently. To maintain compatibility with the Printing Manager, you should follow these guidelines:

- Do not test for the contents of undocumented fields.
- Do not set fields in the `TPrint` record directly.
- Use the print dialog boxes provided by the printer drivers or, if you want to customize these dialog boxes, alter them only as recommended in "Altering the Style or Job Dialog Box" on page 9-35.

Printing a Document

When writing an application, the code you provide that handles printing is referred to as the *printing loop*. A printing loop calls all the Printing Manager routines necessary to print a document. In general, a printing loop must do the following tasks:

- It must unload unused code segments to ensure that you have as much memory as possible in which to print.
- It must open the Printing Manager and the current printer driver by using the `PrOpen` procedure.
- It must set up a valid `TPrint` record for the document (using any values the user previously specified through the style dialog box) by using the `PrintDefault` procedure or the `PrValidate` function. (When the user is printing from the Finder, it is best not to display the style dialog box, but rather to use saved or default settings for the document.)

- It must display the job dialog box as appropriate by using the `PrJobDialog` function or, for a customized job dialog box, the `PrDlgMain` function. (When the user is printing from the Finder, display the job dialog box only once, and then use the `PrJobMerge` procedure to apply the information from this dialog box to any other documents selected by the user.)
- It must determine the number of pages required to print the requested range of pages by examining the fields of the `TPrint` record. (Depending on the page rectangle of the current printer, the amount of data you can fit on a physical page of paper may differ from that displayed on the screen, although it is usually the same.)
- It must determine the number of copies to print by examining the `TPrint` record.
- It may display a status dialog box indicating to the user the status of the current printing operation by using the Dialog Manager function `GetNewDialog` (described in the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*).
- If it displays a status dialog box, it must install an idle procedure in the `pIdleProc` field of the `TPrJob` record (which is contained in the `TPrint` record) to update information in the status dialog box and to check whether the user wants to cancel the operation. (The default idle procedure also performs this check, but if you update information in your status dialog box you must provide your own idle procedure.)
- It must print the requested range of pages for each requested copy by
  - using the `PrOpenDoc` function to open a printing graphics port if the current page number is the first page or a multiple of the value represented by the constant `iPFMaxPgs`
  - opening a page for printing by using the `PrOpenPage` procedure
  - printing the page by drawing into the printing graphics port with the QuickDraw routines described in the rest of this book
  - closing the page by using the `PrClosePage` procedure
  - using the `PrCloseDoc` procedure to close the printing graphics port and begin printing the requested range of pages
  - checking whether the printer driver is using deferred printing and, if so, using the `PrPicFile` procedure to send the spool file to the printer
- Finally, the printing loop must close the Printing Manager by using the `PrClose` procedure.

Listing 9-2 shows an extremely broad example of a printing loop—the code does not optimize for the type of printer being used or for the material being printed (text, graphics, or a mixture of both). However, this sample routine, called `MyPrintLoop`, does cover the major aspects of a printing loop: how to balance calls to the open and close routines, how to determine page count, and how to provide support for documents exceeding the maximum page length specified by the constant `iPFMaxPgs`.

Listing 9-2 A sample printing loop

```
PROCEDURE MyPrintLoop(docToPrint: MyDocRecHnd; displayJob: Boolean);
VAR
   copies, numberOfCopies:    Integer;
   firstPage, lastPage:       Integer;
   pageNumber, numberOfPages: Integer;
   doPrint, changed:          Boolean;
   oldPort:                   GrafPtr;
   theStatus:                 TPrStatus;
   printError:                Integer;
BEGIN
   GetPort(oldPort);
   MyUnLoadTheWorld; {swap out those segments of code not needed to print}
   PrOpen;           {open Printing Manager and the current printer driver}
   IF (PrError = noErr) THEN
   BEGIN
      gPrintResFile := CurResFile;     {save the current resource file}
      gPrintRec := docToPrint^^.docPrintRecHdl; {set to this doc's print rec}
      changed := PrValidate(gPrintRec);   {verify TPrint record}
      IF (PrError = noErr) THEN
      BEGIN
         {determine the number of pages required to print the document}
         numberOfPages := MyDetermineNumOfPages(gPrintRec^^.prInfo.rPage);
         {display job dialog box if requested, else use previous settings}
         IF displayJob THEN
            doPrint := PrJobDialog(gPrintRec)
         ELSE
            doPrint := MyDoJobMerge(gPrintRec);
         IF doPrint THEN
         BEGIN
            numberOfCopies := gPrintRec^^.prJob.iCopies;
            firstPage := gPrintRec^^.prJob.iFstPage;{save first page number}
            lastPage := gPrintRec^^.prJob.iLstPage; {save last page number}
            gPrintRec^^.prJob.iFstPage := 1;        {reset to 1}
            gPrintRec^^.prJob.iLstPage := iPrPgMax; {reset to maximum}
```

```
IF (numberOfPages < lastPage) THEN
   lastPage := numberOfPages;    {to prevent printing past last }
                                 { page}
{display a "Print Status" dialog box (optional)-- }
{ first, deactivate front window}
MyDoActivateFrontWindow(FALSE, oldPort);
gPrintStatusDlg := GetNewDialog(kPrintStatus, NIL, Pointer(-1));
{set up dialog items (insert name of document being printed)}
MySetUpDBoxItems(docToPrint);
ShowWindow(gPrintStatusDlg);  {display the dialog box}
{set up idle procedure (for later use)}
gPrintRec^^.prJob.pIdleProc := @MyDoPrintIdle;
{print the requested number of copies}
FOR copies := 1 TO numberOfCopies DO
BEGIN
   UseResFile(gPrintResFile);{restore driver's resource file}
   {print the requested range of pages of the document}
   FOR pageNumber := firstPage TO lastPage DO
   BEGIN
      {check current page number against iPFMaxPgs}
      IF (pageNumber - firstPage) MOD iPFMaxPgs = 0 THEN
      BEGIN
         IF pageNumber <> firstPage THEN
         {if max size of spool file has been reached (and this }
         { isn't the first page), then close the document, }
         { initiate printing, then reopen the document}
         BEGIN
            PrCloseDoc(gPrinterPort);
            {next line tests for deferred printing}
            IF (gPrintRec^^.prJob.bJDocLoop = bSpoolLoop)
                  AND (PrError = noErr) THEN
               PrPicFile(gPrintRec, NIL, NIL, NIL, theStatus);
         END;
         {if this is the first page or a multiple of iPFMaxPgs, }
         { then open the document for printing}
         gPrinterPort := PrOpenDoc(gPrintRec, NIL, NIL);
      END; {of check current page number}
      IF (PrError = noErr) THEN
      BEGIN {print a page}
         PrOpenPage(gPrinterPort, NIL);
         IF (PrError = noErr) THEN
         {draw (print) a page in the printable area for the }
         { current printer (indicated by the rPage field)}
```

```
                MyDrawStuff (gPrintRec^^.prInfo.rPage, docToPrint,
                             GrafPtr(gPrinterPort), pageNumber);
                PrClosePage(gPrinterPort);
             END;  {of print a page}
          END; {of print the requested range of pages}
          PrCloseDoc(gPrinterPort);
          IF (gPrintRec^^.prJob.bJDocLoop = bSpoolLoop) AND
             (PrError = noErr) THEN
             PrPicFile(gPrintRec, NIL, NIL, NIL, theStatus);
       END;
    END;
 END;
 END;
 printError := PrError;
 PrClose;
 IF (printError <> noErr) THEN
    DoError(ePrint, printError);
 DisposeDialog(gPrintStatusDlg);
 SetPort(oldPort);
 MyDoActivateFrontWindow(TRUE, oldPort); {activate window}
END;
```

The `MyPrintLoop` procedure starts by getting a pointer to the current graphics port. Then it calls an application-defined routine, `MyUnloadTheWorld`, that swaps out code segments not required during printing. Then it opens the Printing Manager and the current printer driver and its resource file by calling `PrOpen`.

The `MyPrintLoop` procedure saves the current resource file (after calling `PrOpen`, the current resource file is the driver's resource file) so that, if its idle procedure changes the resource chain in any way, it can restore the current resource file before returning; thus the driver does not lose access to its resources. The `MyPrintLoop` procedure then uses the `PrValidate` function to change any values in the `TPrint` record associated with the document to match those specified by the current printer driver; these values can be changed later by the printer driver as a result of your application's use of the `PrStlDialog` and `PrJobDialog` functions. (Your application passes a handle to a `TPrint` record to the `PrStlDialog` and `PrJobDialog` functions, and these procedures modify the `TPrint` record according to the user's interaction with the style and job dialog boxes.) The `MyPrintLoop` procedure calls `PrValidate` rather than `PrintDefault` to preserve any values that the user might have previously set through the style dialog box.

To print a document, you must divide the data into sections that fit within the page rectangle dimensions stored in the `rPage` field of the `TPrJob` record, which is contained in the `TPrint` record. (This information is stored in the `rPage` field when you call the `PrintDefault`, `PrValidate`, or `PrStlDialog` routine.) The application-defined function `MyDetermineNumOfPages` is specific to the application, because the way the application divides up the data depends on the type of text and graphics in the document. The `MyDetermineNumOfPages` function determines the number of pages required to print the document by comparing the size of the document with the printable area for the current printer, which is specified by the value in the `rPage` field of the `TPrJob` record in the `TPrint` record.

After determining the number of pages required to print the document, `MyPrintLoop` displays the job dialog box if the calling routine requested it to do so. If the user prints multiple documents at once, the calling routine sets the `displayJob` parameter to `TRUE` for the first document and `FALSE` for subsequent documents. This allows the user to specify values in the job dialog box only once when printing multiple documents. It also provides an application with the ability to print documents in the background (for example, as the result of responding to the Apple event Print Documents) without requiring the application to display the job dialog box.

The user's responses in the job dialog box provide such information as the number of copies and the page numbers of the first and last pages requested. The `MyPrintLoop` procedure stores these values in the local variables `firstPage` and `lastPage`. It then resets the value of the first page in the `TPrJob` record as 1 and resets the value of the last page to the value represented by the constant `iPrPgMax`.

The `MyPrintLoop` procedure compares the values of the number of pages in the document with the last page the user requested and changes the last page number as necessary. For example, if the user asks to print page 50 of a two-page document, `MyPrintLoop` resets the value of the last page to 2.

At this point, `MyPrintLoop` is about to begin the process of sending the pages off to be printed. So it displays its own status dialog box to inform the user of the current status of the printing operation. If your status dialog box provides a button or reports on the progress of the printing operation, you need to handle events in the dialog box by providing an idle procedure. Your idle procedure should update the items in your status dialog box to show the current progress of the printing operation, and it should determine whether the user has canceled the printing operation. The printer driver calls the idle procedure periodically during the printing process. For more information on idle procedures, see "Writing an Idle Procedure" on page 9-38.

After installing its idle procedure, the `MyPrintLoop` procedure then begins the printing operation by performing a number of steps for each requested copy. First, `MyPrintLoop` restores the current resource file to the printer driver's resource file.

`MyPrintLoop` then begins the process of printing each page. The maximum number of pages that can be printed at a time is represented by the constant `iPFMaxPgs`. If the file is larger than the value represented by `iPFMaxPgs`, your application can print the number of pages represented by `iPFMaxPgs` and then begin the printing loop again with the next section of the document. In this way, you can print any number of pages.

Next, `MyPrintLoop` opens a page for printing and draws the page in the printing graphics port with the application-defined `MyDrawStuff` procedure, the details of which are specific to the application. The parameters to `MyDrawStuff` are the size of the page rectangle, the document containing the data to print, the printing graphics port in which to draw, and the page number to be printed. This allows the application to use the same code to print a page of a document as it uses to draw the same page on screen.

When `MyPrintLoop` is finished printing (or has printed a multiple of the value represented by the constant `iPFMaxPgs`), it closes the printing graphics port for the document. By testing for the `bSpoolLoop` constant in the `bJDocLoop` field of the `TPrJob` record, `MyPrintLoop` determines whether a printer driver is using deferred printing; if so, `MyPrintLoop` calls the `PrPicFile` procedure, which sends the spool file to the printer.

Some QuickDraw printer drivers (in particular, those for the ImageWriter and ImageWriter LQ printers) provide two methods of printing documents: deferred and draft-quality. Typically, the printer driver uses deferred printing when a user chooses Best in the job dialog box, and it uses draft-quality printing when the user chooses Draft.

Deferred printing was designed to allow ImageWriter printers to spool a page image to disk when printing under the low memory conditions of the original 128 KB Macintosh computer. With **deferred printing,** a printer driver records each page of the document's printed image in a structure similar to a QuickDraw picture, which the driver writes to a spool file. For compatibility with printer drivers that still support deferred printing, use the `PrPicFile` procedure to instruct these printer drivers to turn the QuickDraw pictures into bit images and send them to the printer. (**Draft-quality printing,** on the other hand, is a method by which a printer driver converts into drawing operations calls only to QuickDraw's text-drawing routines. The printer driver sends these routines directly to the printer instead of using deferred printing to capture the entire image for a page in a spool file.)

Note

Do not confuse background printing with deferred printing. While printer drivers supporting background printing also create spool files, you do not need to use the `PrPicFile` procedure to send these spool files to the printer. In fact, there is no reliable way for you to determine whether a printer driver is using a spool file for background printing. ◆

The `MyPrintLoop` procedure concludes by closing the Printing Manager, reporting any Printing Manager errors, and resetting the current graphics port to the original port.

In your printing loop, you should balance all calls to Printing Manager open routines to the equivalent Printing Manager close routines. This is extremely important, even if you stop printing because of an error. Failure to call the matching close routines can cause the Printing Manager to perform incorrectly.

Note that `MyPrintLoop` calls `PrError` after each Printing Manager routine. If an error is found, the loop calls a close routine (`PrClose`, `PrClosePage`, or `PrCloseDoc`) for any Printing Manager open routines (`PrOpen`, `PrClosePage`, or `PrOpenDoc`) before informing the user of the error. You should use this approach in your own application to make sure the Printing Manager closes properly and all temporary memory is released.

▲ **WARNING**
Some applications use a method of printing that prints out each page of a spooled document as a separate print job in order to avoid running out of disk space while spooling the document. You should not use this method, known as "spool a page, print a page." It is appropriate only for a printer directly connected to the user's computer (that is, not to a network) and therefore creates device dependence—and also it's extremely slow. If the printer is a remote or shared device (such as a LaserWriter printer connected by an AppleTalk network), another application could print a document between the pages of your user's document. At worst, if both applications printing to the shared printer use the "spool a page, print a page" method, the printed documents could end up interleaved. The pages for one of the documents could be out of order, even when printed by itself on a shared, network printer. ▲

Printing From the Finder

Typically, users print documents that are open on the screen one at a time while the application that created the document is running. Alternatively, users can print one or more documents from the Finder. To print documents from the Finder, the user selects one or more document icons and chooses the Print command from the File menu. When the Print command is chosen, the Finder starts up the application and passes it an Apple event—the Print Documents event—indicating that the documents are to be printed rather than opened on the screen.

As explained in *Inside Macintosh: Interapplication Communication*, your application should support the required Apple events, which include the Print Documents event. In response to a Print Documents event, your application should do the following:

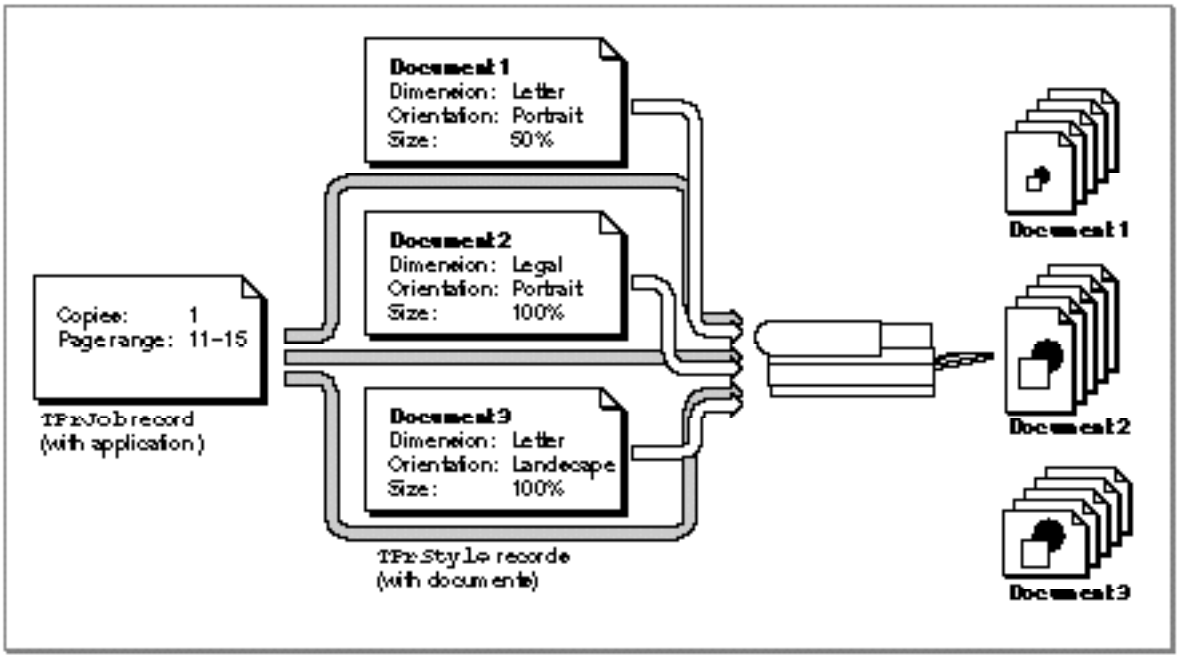
1. Your application should not open windows for the documents.
2. For style information, your application should use saved or default settings instead of displaying the style dialog box to ask this information from the user.

3. Your application should use the `PrJobDialog` function (described on page 9-62) or the `PrDlgMain` function (described on page 9-63) to display the job dialog box only once. When the user clicks the OK button in the job dialog box, you can then use the `PrJobMerge` procedure (described on page 9-66) to apply the information specified by the user to all of the documents selected from the Finder.

   For example, if the user has selected three documents to print, you can display the job dialog box only once and then apply the same information supplied by the user to all three documents. Figure 9-10 shows a situation where, through the job dialog box, the user has specified the number of copies and the range of pages to print. In this example, the application applies this job information to the `TPrint` record of the three documents by calling `PrJobMerge`. Note that `PrJobMerge` preserves the fields of the `TPrint` record that are specific to each document (that is, the fields that are set by the user through the style dialog box).

4. Your application should remain open until the Finder sends your application a Quit event. If appropriate, the Finder sends your application this Apple event immediately after sending it the Print Documents event.

Figure 9-10 How the `PrJobMerge` procedure works



See *Inside Macintosh: Interapplication Communication* for more information about how to handle the Print Documents and Quit events.

Providing Names of Documents Being Printed

Some printer drivers (usually those for printers such as LaserWriter printers that are shared among many users) provide the names of the users who are printing and the documents that are being printed to others interested in using the printer. Providing the names of users and documents is a courtesy to other users sharing the printer on a network. The printer driver gets the name of the document being printed from the title of the frontmost window on the user's screen. The `PrOpenDoc` and `PrValidate` functions call the Window Manager procedure `FrontWindow` to get the document's name.

Printer drivers can't get a document name if your application doesn't display windows while printing. For example, applications should not open windows for their documents when the user prints from the Finder. If there is no front window, or if the window's title is empty, the printer driver sets the document name to "Unspecified" or "Untitled."

You can ensure that the document name is available by displaying a printing status dialog box and setting the window's title to the document's name. If the dialog box is one that doesn't have a title bar (like that of type `dBoxProc`), this title is not displayed but the current printer driver can still use the title as the document's name. If you don't want to put up a visible window, you can create a tiny window (for instance, type `plainDBox`) and hide it behind the menu bar by giving it the global coordinates of (1,1,2,2). See the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about the `dBoxProc` and `plainDBox` window types.

Note

Do not set the document name in the `TPrint` record directly. Not all printer drivers support this field, and Apple does not guarantee that internal fields of the Printing Manager's data structures will remain the same. ◆

Printing Hints

QuickDraw is the primary means you use to print, and in general you can use QuickDraw in the printing graphics port exactly as you would for a screen's graphics port. There are a few things to note when drawing to the printing graphics port:

- Don't depend on values in a printing graphics port remaining identical from page to page. With each new page, you usually get reinitialized font information and other characteristics for the printing graphics port.
- Don't make calls that don't do anything on the printer. For example, QuickDraw erase routines such as `EraseRect` are quite time-consuming and normally aren't needed on the printer. An erase routine takes time because every bit (90,000 bits per square inch on a 300 dpi LaserWriter) has to be cleared. Paper does not need to be erased the way the screen does. Also avoid using the `TextBox` procedure, which makes calls to the `EraseRect` procedure. You might want to use a different method of displaying text (for example, `DrawString` or `DrawText`) or write your own version of `TextBox`. See the chapter "QuickDraw Text" in *Inside Macintosh: Text*.

- Don't use clipping to select text to be printed. There are a number of subtle differences between how text appears on the screen and how it appears on the printer; you can't count on knowing the exact dimensions of the rectangle occupied by the text.
- Don't use fixed-width fonts to align columns. Because spacing is adjusted on the printer, you should explicitly move the pen to where you want it.
- Don't use the outline font style to create white text on a black background.
- Avoid changing fonts frequently.
- Because of the way rectangle intersections are determined, you slow printing substantially if your clipping region falls outside of the rectangle given by the `rPage` field of the `TPrInfo` record of the `TPrint` record.

Getting and Setting Printer Information

You can determine the resolution of the printer, set the resolution you want, find out if the user has selected landscape printing, or force enhanced draft-quality printing by using the `PrGeneral` procedure. You call the `PrGeneral` procedure with one of five opcodes: `getRslDataOp`, `setRslOp`, `getRotnOp`, `draftBitsOp`, or `noDraftBitsOp`. These opcodes have data structures associated with them.

When you call the `PrGeneral` procedure, it in turn calls the current printer driver to get or set the desired information. Not all printer drivers support all features provided by the `PrGeneral` procedure, however, so your application can't depend on its use.

Listing 9-3 shows an application-defined routine, `DoIsPrGeneralThere`, that checks whether the current printer driver supports the `PrGeneral` procedure. First, `DoIsPrGeneralThere` sets the `opcode` field of the `TGetRotnBlk` record to the `getRotnOp` opcode—the opcode used to determine whether the user has chosen landscape orientation. Then `DoIsPrGeneralThere` passes the address of the `TGetRotnBlk` record to the `PrGeneral` procedure. It then calls `PrError` to get any errors that result from calling `PrGeneral`. If the error is `resNotFound`, the printer driver does not support `PrGeneral`.

Listing 9-3 Checking whether the current printer driver supports the `PrGeneral` procedure

```
FUNCTION DoIsPrGeneralThere: Boolean;
VAR
   getRotRec:     TGetRotnBlk;
   myPrintErr:    OSErr;
BEGIN
   myPrintErr := 0;
   getRotRec.iOpCode := getRotnOp;  {set the opcode}
   getRotRec.hPrint := gMyPrRecHdl; {TPrint record this operation applies to}
   PrGeneral(@getRotRec);
   myPrintErr := PrError;
   PrSetError(noErr);
   IF (myPrintErr = resNotFound) THEN  {the current driver doesn't support }
      DoIsPrGeneralThere := FALSE;     { PrGeneral}
   ELSE
      DoIsPrGeneralThere := TRUE;      {current driver supports PrGeneral}
END;
```

After determining that the current printer driver supports `PrGeneral`, you can use `PrGeneral` to

- determine and set the resolution of the current printer
- determine the current page orientation
- force enhanced draft-quality printing

As an alternative to testing for `PrGeneral`, your application can call `PrGeneral` and then test whether `PrError` error returns the `opNotImpl` result code, which indicates that the printer driver either does not support `PrGeneral` or does not support that particular opcode.

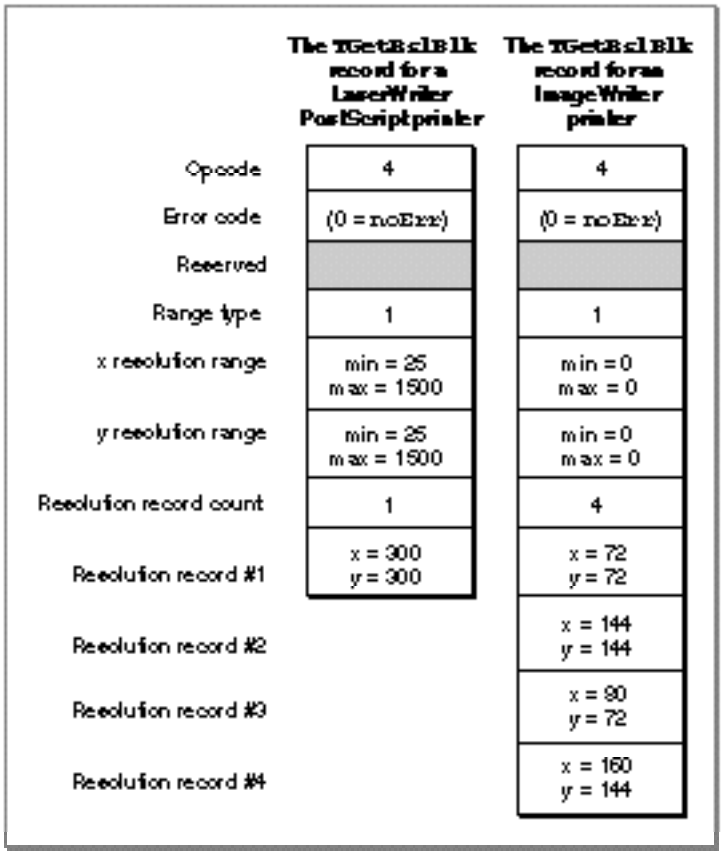
These operations are discussed in the following sections.

Determining and Setting the Resolution of the Current Printer

Some printer drivers support only one of the two possible kinds of resolution: discrete or variable. You can use the `PrGeneral` procedure to determine the kind of resolution supported by the current printer and then use the highest resolution desired by your application or the user.

Each printer has its own imaging capabilities. When you call `PrGeneral` with the value `getRslDataOp` in the `iOpCode` field of the `TGetRslBlk` record, `PrGeneral` returns the resolutions that the printer supports. Figure 9-11 shows `TGetRslBlk` records (described on page 9-53) returned by the drivers for a 300-dpi LaserWriter PostScript printer and a QuickDraw ImageWriter printer. Because it supports variable resolutions, the `TGetRslBlk` record for the LaserWriter driver specifies minimum and maximum resolutions in the x and y directions. Because it uses discrete resolutions, the `TGetRslBlk` record for the ImageWriter driver specifies no minimum or maximum resolutions in the x and y directions, but instead specifies the four discrete resolutions it supports.

Figure 9-11 Sample resolutions for a PostScript printer and a QuickDraw printer



A TPrint record contains the x and y resolutions that the printer uses in printing the data associated with the TPrint record. For each TPrint record you use, you can either use the default values or you can specify the particular imaging resolution that you want to use. To do this, you can call PrGeneral, specifying the value setRslOp in the iOpCode field and specifying the x and y resolutions in the iXRsl and iYRsl fields of the TSetRslBlk record (which is described on page 9-54). The PrGeneral procedure returns the noErr result code if it has updated the TPrint record with this new resolution, or it returns the noSuchRsl result code if the current printer doesn't support this resolution.

Listing 9-4 illustrates how to use the PrGeneral procedure to determine the possible resolutions for the current printer and then set a TPrint record to the desired resolution.

Listing 9-4 Using the getRslDataOp and setRslOp opcodes with the PrGeneral procedure

```
FUNCTION DoSetMaxResolution (thePrRecHdl: THPrint): Integer;
VAR
   maxDPI:     Integer;
   resIndex:   Integer;
   getResRec:  TGetRslBlk;
   setResRec:  TSetRslBlk;
BEGIN
   maxDPI := 0;
   getResRec.iOpCode := getRslDataOp;{get printer resolution info}
   PrGeneral(@getResRec);
   IF (getResRec.iError = noErr) AND (PrError = noErr) THEN
   BEGIN
      {the TGetRslBlk record contains an array of possible resolutions-- }
      { so loop through each resolution range record looking for }
      { the highest resolution available where x and y are equal}
      FOR resIndex := 1 TO (getResRec.iRslRecCnt) DO
      BEGIN
         IF (getResRec.rgRslRec[resIndex].iXRsl =
               getResRec.rgRslRec[resIndex].iYRsl) AND
               (getResRec.rgRslRec[resIndex].iXRsl > maxDPI) THEN
            maxDPI := getResRec.rgRslRec[resIndex].iYRsl;
      END;
      {set the resolution to the maximum supported resolution}
      IF maxDPI <> 0 THEN
      BEGIN
         WITH setResRec DO
         BEGIN
            iOpCode := setRslOp;
            hPrint := thePrRecHdl;
```

```
            iXRsl := maxDPI;
            iYRsl := maxDPI;
         END;
         PrGeneral(@setResRec);
      END; {end of maxDPI <> 0}
      IF (setResRec.iError = noErr) AND (PrError = noErr) AND
            (maxDPI <> 0) THEN
         DoSetMaxResolution := maxDPI;
   END
   ELSE
      DoSetMaxResolution := 0;
END;
```

You can reset the original resolutions by calling the `PrGeneral` procedure with the `setRslOp` opcode a second time. To do so, you should save the values contained in the `iVRes` and `iHRes` fields of the `TPrInfo` record before making the first call to `PrGeneral`. You can also reset the original resolutions by calling the `PrintDefault` procedure with the `TPrint` record, which sets all of the fields of the `TPrint` record to the default values of the current printer resource file. However, if you use `PrintDefault` you lose all of the user's selections from the last style dialog box. (You may want to reset the original resolution because that may be the printer's best resolution, though not its highest.)

Based on the information you get with a call to `PrGeneral` using the `getRslDataOp` opcode, you may decide to change the resolution with a call to `PrGeneral` using the `setRslOp` opcode. If so, the printer driver may need to change the appearance of the style and job dialog boxes by disabling some items. Therefore, you should determine and set the resolution *before* you use the `PrStlDialog` and `PrJobDialog` functions (or the `PrDlgMain` function) to present the print dialog boxes to the user.

Note that the style dialog boxes for some printers, such as the StyleWriter, may offer the user a choice of printing in Best or Normal modes, which sets the printing at 360 or 180 dpi, respectively. Your application has no control over this setting. The printer driver converts your drawing accordingly.

Determining Page Orientation

At times it can be useful for your application to determine which page orientation the user selects in the style dialog box. For instance, if an image fits on a page only if it is printed in landscape orientation (the `prInfo` field of the `TPrint` record defines a smaller horizontal value for the paper rectangle than for the image rectangle) and the user has not selected landscape orientation, your application can remind the user to select this orientation before printing. Otherwise, the user gets a clipped image.

If you call the `PrGeneral` procedure with the `getRotnOp` opcode in the `TGetRotnBlk` record (described on page 9-56), the printer driver returns in the `fLandscape` field of this record a Boolean variable that indicates whether or not the `TPrint` record specifies

landscape orientation. The user selects the type of orientation through the style dialog box, and the printer driver updates the fields of the `TPrint` record accordingly.

Listing 9-5 shows an application-defined function, `DoIsLandscapeModeSet`, that returns a Boolean value indicating whether the user has selected landscape orientation for the current document.

Listing 9-5 Using the `getRotnOp` opcode with the `PrGeneral` procedure to determine page orientation

```
FUNCTION DoIsLandscapeModeSet (thePrRecHdl: THPrint): Boolean;
VAR
   getRotRec:  TGetRotnBlk;
BEGIN
   getRotRec.iOpCode := getRotnOp;  {set opcode}
   getRotRec.hPrint := thePrRecHdl; {specify TPrint record}
   PrGeneral(@getRotRec);           {get landscape orientation}
   IF (getRotRec.iError = noErr) AND (PrError = noErr) AND
      getRotRec.fLandscape THEN
      DoIsLandscapeModeSet := TRUE
   ELSE
      DoIsLandscapeModeSet := FALSE;
END;
```


Enhancing Draft-Quality Printing

When the user selects faster, draft-quality printing from a job dialog box from some printer drivers, the printer driver handles the printing operation appropriately.

However, you can force users to use an enhanced form of draft-quality printing on ImageWriter printers (as well as on other printers that may support enhanced draft-quality printing) by calling the `PrGeneral` procedure, specifying the `draftBitsOp` opcode in a `TDftBitsBlk` record (described on page 9-55), and specifying the `TPrint` record for the operation. If your application produces only text, bitmaps, or pixel maps, this can increase performance and save disk space, because the printer driver prints the document immediately, rather than spooling it to disk as with deferred printing. The `draftBitsOp` opcode has no effect if the printer driver does not support draft-quality printing or does not support deferred printing. If the driver does not support the `draftBitsOp` opcode, the `PrGeneral` procedure returns the `opNotImpl` result code.

With draft-quality printing, a printer driver like the ImageWriter printer driver converts into drawing operations calls only to QuickDraw's text-drawing routines. The printer driver sends these text-drawing routines directly to the printer instead of using deferred printing to capture the entire image for a page in a spool file. Draft-quality printing produces quick, low-quality drafts of text documents that are printed straight down the page, from top to bottom and left to right.

Using the `PrGeneral` procedure, it's possible to produce enhanced draft-quality printing on some printers—such as ImageWriter printers. Normally, draft-quality printing renders output consisting only of text. However, **enhanced draft-quality printing** prints the bitmaps and pixel maps that your application draws using the `CopyBits` procedure (described in the chapter "QuickDraw Drawing" in this book) without using deferred printing to write to and read from a spool file.

Because it's supported by so few printer drivers, and because it offers little in the way of extra capability, enhanced draft-quality printing has limited usefulness.

To use enhanced draft-quality printing, call `PrGeneral` with the `draftBitsOp` opcode before using the `PrStlDialog` and `PrJobDialog` functions or the `PrDlgMain` function to present the style dialog box and job dialog box to the user. The use of the `draftBitsOp` opcode may cause items in the print dialog boxes to become inactive. For the ImageWriter printer driver, for example, the use of the `draftBitsOp` opcode makes the landscape icon in the style dialog box and the Best and Faster options in the job dialog box inactive.

IMPORTANT

If you call `PrGeneral` with the `draftBitsOp` opcode after using the `PrJobDialog` or `PrDlgMain` function, and if the user chooses draft printing from the job dialog box, the ImageWriter printer does not print any bitmaps or pixel maps contained in the document. ▲

Listing 9-6 illustrates how to implement enhanced draft-quality printing.

Listing 9-6 Using the `draftBitsOp` opcode with the `PrGeneral` procedure for enhanced draft-quality printing

```
FUNCTION DoDraftBits (thePrRecHdl: THPrint): Boolean;
VAR
   draftBitsBlk:  TDftBitsBlk;
BEGIN
   draftBitsBlk.iOpCode := draftBitsOp;{set the opcode}
   draftBitsBlk.hPrint := thePrRecHdl; {specify the TPrint record}
   PrGeneral(@draftBitsBlk);        {use enhanced draft quality}
   IF (draftBitsBlk.iError = noErr) AND (PrError = noErr) THEN
      DoDraftBits := TRUE     {this TPrint record specifies }
                              { enhanced draft printing}
   ELSE
      DoDraftBits := FALSE;   {this TPrint record does not }
                              { specify enhanced draft printing}
END;
```

You should keep one additional point in mind when using the `draftBitsOp` opcode: all of the data that is printed must be sorted along the y axis, because reverse paper motion is not possible on the ImageWriter printer when printing in draft-quality mode. This means that you cannot print two objects side by side; that is, the top boundary of an object cannot be higher than the bottom boundary of the previous object. To get around this restriction, you should sort your objects before print time.

You can call `PrGeneral` with the `noDraftBitsOp` opcode to use regular draft-quality printing again. If you call `PrGeneral` with `noDraftBitsOp` without first calling `draftBitsOp`, the procedure does nothing. As with the `draftBitsOp` opcode, you should call `PrGeneral` with the `noDraftBitsOp` opcode before you present the style and job dialog boxes to the user.

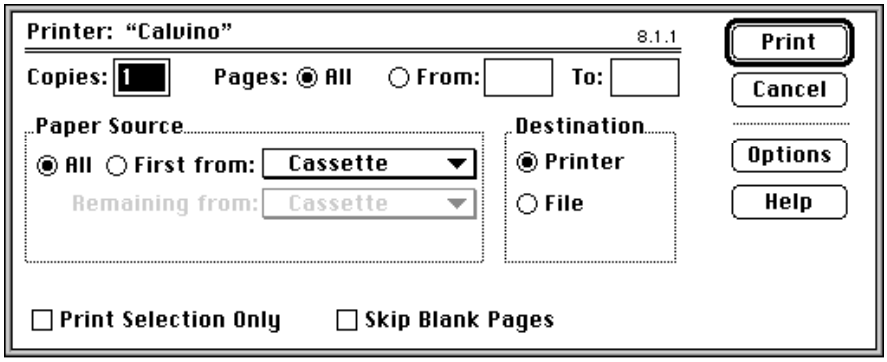
Altering the Style or Job Dialog Box

Each printer resource file includes definitions of the standard style and job dialog boxes that are specific to the type of printer managed by its printer driver. The `PrStlDialog` and `PrJobDialog` functions display the style and job dialog boxes defined by the resource file of the current printer.

For example, the standard style and job dialog boxes for the LaserWriter printer driver are shown in Figure 9-3 on page 9-7 and Figure 9-5 on page 9-8, respectively. The standard dialog boxes provided by the StyleWriter printer driver are shown in Figure 9-2 on page 9-6 and Figure 9-4 on page 9-7. Each dialog box has options that the user can set. If you want to use the standard style or job dialog box provided by the printer driver for the current printer, call the `PrStlDialog` function or the `PrJobDialog` function.

You may wish to add some additional options to these dialog boxes so that the user can customize the printing process even further. For example, Figure 9-12 illustrates a print job dialog box with two additional checkboxes: Print Selection Only and Skip Blank Pages.

Figure 9-12 A print job dialog box with additional checkboxes



You must follow these guidelines if you alter the style or job dialog boxes:

- Add additional options below the standard ones in the dialog box and don't change the standard ones—that is, don't delete, rearrange, or add new items in the existing list.
- Don't count on an item retaining its current position on the screen or in the dialog item list.
- Don't use more than half the smallest screen height for your items. (The smallest screen height is the 9-inch Macintosh Classic screen.) Printer drivers are allowed to expand the items in the standard dialog boxes to fill the top half of a 9-inch screen.
- If you want to add a lot of items to the dialog boxes, be aware this may confuse users. You should consider having your own separate dialog box in addition to the existing style and job dialog boxes.

You can customize a style or job dialog box by undertaking the following steps:

- Use the `PrOpen` procedure to open the Printing Manager.
- Use the `PrStlInit` or `PrJobInit` function to initialize a `TPrDlg` record. This record, described on page 9-50, contains the information needed to set up the style or job dialog box.
- Define an initialization routine that appends items to the printer driver's style or job dialog box. Your initialization routine should
  - use the Dialog Manager procedure `AppendDITL` to add items to the dialog box whose `TPrDlg` record you initialized with `PrStlInit` or `PrJobInit`
  - install two functions in the `TPrDlg` record: one—in the `pFltrProc` field—for handling events (such as update events for background applications) that the Dialog Manager doesn't handle in a modal dialog box, and another—in the `pItemProc` field—for handling events in the items added to the dialog box (for example, when the user clicks a checkbox that your application adds)
  - return a pointer to the `TPrDlg` record
- Pass the address of your initialization routine to the `PrDlgMain` function to display the dialog box.
- Respond to the dialog box as appropriate.
- Use the `PrClose` procedure when you are finished using the Printing Manager.

The event filter function pointed to in the `pFltrProc` field of the `TPrDlg` record extends the Dialog Manager's ability to handle events. When your application displays the style or job dialog box, you can use an event filter function to handle events that the Dialog Manager doesn't handle for modal dialog boxes. The chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes how to write an event filter function for the Dialog Manager.

The routine you supply in the `pItemProc` field of the `TPrDlg` record should handle events in the items you add to the dialog box. Sometimes called a *dialog hook*, this routine typically responds to clicks in the radio buttons or checkboxes that you add to the dialog box.

Listing 9-7 shows an application-defined routine called `DoPrintDialog` that specifies its own initialization function, called `MyPrDialogAppend`.

Listing 9-7 Installing an initialization function to alter the print job dialog box

```
FUNCTION DoPrintDialog: OSErr;   {display print job dialog box}
BEGIN
   PrOpen;     {open the Printing Manager}
   gPrintRec := THPrint(NewHandle(sizeof(TPrint)));{create a TPrint record}
   PrintDefault(gPrintRec);   {use default values for the TPrint record}
   gPrJobDialogBox := PrJobInit(gPrintRec);  {get a pointer to the }
                                             { invisible job dialog box}
   {use PrDlgMain to display the altered job dialog box}
   IF (PrDlgMain(gPrintRec, @MyPrDialogAppend)) THEN
      MyPrintDoc;
   PrClose;    {close the Printing Manager}
END;
```

The application-defined routine `MyPrDialogAppend` is shown in Listing 9-8. It uses the Resource Manager function `GetResource` to get a handle to an item list (`'DITL'`) resource containing the two extra checkboxes shown in Figure 9-12 on page 9-35. Using the Dialog Manager procedure `AppendDITL`, `MyPrDialogAppend` appends the items in this item list resource to the print job dialog box. Then `MyPrDialogAppend` installs the application's event filter function for modal dialog boxes. Finally, `MyPrDialogAppend` installs it own routine, called `HandleMyAppendedItems`, to handle clicks in the two newly installed checkboxes.

Listing 9-8 Adding items to a print job dialog box

```
FUNCTION MyPrDialogAppend (hPrint: THPrint): TPPrDlg;
VAR
  MyAppendDITLH: Handle;
BEGIN
   IF gDITLAppended = FALSE THEN
   BEGIN
      {first, get item list resource containing checkboxes}
      MyAppendDITLH := GetResource('DITL', kPrintingCheckBoxes);
```

```
      {next, append this item list resource to job dialog box}
      AppendDITL(DialogPtr(gPrJobDialogBox), MyAppendDITLH,
                 appendDITLBottom);
      gDITLAppended := TRUE;
   END;
   gFltrItemProc := LongInt(gPrJobDialogBox^.pFltrProc);
   {put an event filter function (to handle events that Dialog }
   { Manager doesn't handle in modal dialog boxes) }
   { in the pFltrProc field of the TPrDlg record}
   gPrJobDialogBox^.pFltrProc := ProcPtr(@MyEventFilter);
   gPrItemProc := LongInt(gPrJobDialogBox^.pItemProc);
   {put a dialog hook to handle clicks in appended items }
   { in the pItemProc field of the TPrDlg record}
   gPrJobDialogBox^.pItemProc := ProcPtr(@HandleMyAppendedItems);
   MyPrDialogAppend := gPrJobDialogBox;
END;
```

See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about item list resources, event filter functions for modal dialog boxes, and the `AppendDITL` procedure. See the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox* for information about the `GetResource` function.

Writing an Idle Procedure

The printer driver for the current printer periodically calls an idle procedure while it sends a document to the printer. The `TPrJob` record contained in the `TPrint` record contains a pointer to an idle procedure in the `pIdleProc` field. If this field contains the value `NIL`, then the printer driver uses the Printing Manager's default idle procedure. The default idle procedure checks for Command-period keyboard events and sets the `iPrAbort` error code if one occurs, so that your application can cancel the print job at the user's request. However, the default idle procedure does not display a print status dialog box. It is up to the printer driver or your application to display a print status dialog box.

Most printer drivers display their own status dialog boxes. However, your application can display its own status dialog box that reports the current status of the printing operation to the user. If it does, your status dialog box should allow the user to press Command-period to cancel the printing operation, and it may also provide a button allowing the user to cancel the printing operation. To handle update events in your status dialog box, Command-period keyboard events, and clicks in your Cancel button (if you provide one), you should provide your own idle procedure. (See Figure 9-9 on page 9-14 for an example of an application-defined status dialog box.)

Here are several guidelines you must follow when writing your own idle procedure.

- If you designate an idle procedure, you must set the `pIdleProc` field of the `TPrJob` record *after* presenting the style and job dialog boxes, validating the `TPrint` record, and initializing the fields in the `TPrint` record (because the routines that perform these operations may reset the `pIdleProc` field to `NIL`). The `TPrJob` record is described on page 9-47.
- You must install your idle procedure in the `TPrint` record before calling the `PrOpenDoc` function. Otherwise, some printer drivers do not give the idle procedure any time to run.
- Do not attempt any printing from within the idle procedure, because the Printing Manager is *not* reentrant.
- Do not reference global variables unless you set up your own A5 world (as described in *Inside Macintosh: Processes*).
- If you use a modal dialog box to display printing status information, you must call the Event Manager function `WaitNextEvent` (described in the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*) to capture mouse events or the Command-period keyboard event that signals that the user wants to cancel printing. Do not call the `WaitNextEvent` function unless you display a modal dialog box.
- So that your application doesn't draw into a printing port, don't call the QuickDraw `OpenPicture` function or the `DrawPicture` procedure from your idle procedure without changing the current graphics port.
- Upon entry to the idle procedure, you must save the printing graphics port, and you must restore it upon exit if you draw anything within the idle procedure. If you don't save and restore the printing graphics port, upon return the printer driver draws into the graphics port of your dialog box instead of its own printing graphics port. To save the printer's printing graphics port, call the `GetPort` procedure when entering the idle procedure. Before you exit, call the `SetPort` procedure to set the port back to the printer driver's printing graphics port. (The `GetPort` and `SetPort` procedures are described in the chapter "Basic QuickDraw" in this book.)
- If your idle procedure changes the resource chain, you should save the reference number of the printer driver's resource file by calling the `CurResFile` function at the beginning of your idle procedure. (Any routine that changes the value of the global variable `TopMapHdl`, such as the `OpenResFile` function or the `UseResFile` procedure, changes the resource chain. Some printer drivers assume the resource chain does not change, and you may get an error if you change it.) When you exit from the idle procedure, restore the resource chain using the `UseResFile` procedure. If you are not changing the resource chain, you do not need to save the resource chain. (The `CurResFile`, `OpenResFile`, and `UseResFile` routines are described in the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.)
- Avoid calling the `PrError` function within the idle procedure. Errors that occur while it is executing are usually temporary and serve only as internal flags for communication within the printer driver, not for the application. If you absolutely must call `PrError` within your idle procedure and an error occurs, do not cancel printing. Wait until the last called printing procedure returns and then check to see if the error still remains.

Listing 9-9 shows an application-defined idle procedure.

Listing 9-9 An idle procedure

```
PROCEDURE MyDoPrintIdle;
VAR
   oldPort:          GrafPtr;
   cursorRgn:        RgnHandle;
   event:            EventRecord;
   gotEvent:         Boolean;
   itemHIt:          Integer;
   handled, canceled:   Boolean;
BEGIN
   GetPort(oldPort);
   SetPort(gPrintStatusDlg);
   cursorRgn := NIL;
   gotEvent := WaitNextEvent(everyEvent, event, 15, cursorRgn);
   IF gotEvent THEN
   BEGIN
      handled := MyStatusHandleEvent(gPrintStatusDlg, event,
                                     itemHit);
      canceled := MyUserDidCancel;
      IF canceled THEN
         itemHit := kStopButton;
      handled := MyDoHandleHitsInStatusDBox(itemHit);
   END;
   MyUpdateStatusInformation(canceled);   {update status }
                               { information in dialog box}
   SetPort(oldPort);
END;
```

The application displays a modal dialog box for its status dialog box, and then installs `MyDoPrintIdle`, which calls the Event Manager function `WaitNextEvent` to capture events while the modal dialog box is displayed.

The `MyDoPrintIdle` procedure first saves the current graphics port (so that it can later restore it) and then sets the current port to the graphics port of the status dialog box. Your idle procedure should save, set, and restore the graphics port in this manner to avoid accidentally drawing in the printing graphics port.

The MyDoPrintIdle procedure then calls WaitNextEvent to get the current event and then calls its own routine to handle the event. (For example, the MyStatusHandleEvent function handles update and activate events.) By calling WaitNextEvent, MyDoPrintIdle gives background applications a chance to handle update events in their windows while the application in this example displays a modal dialog box. (The Dialog Manager does not give background applications a chance to handle update events in their windows when a modal dialog box is displayed.)

MyDoPrintIdle then calls another application-defined procedure to determine whether the user wishes to cancel the printing operation. The MyUserDidCancel function scans the event queue for keyboard events and mouse events. If it finds a Command-period keyboard event, it returns TRUE. The MyUserDidCancel function also returns TRUE if it finds mouse events indicating that the user clicked the Stop Printing button (that is, it uses the Dialog Manager function FindDialogItem to determine whether the mouse location specified in a mouse event is in the Stop Printing button). If the user clicks the Stop Printing button, MyUserDidCancel highlights the button appropriately.

To handle hits in the status dialog box, the MyDoHandleHitsInStatusDBox function simply checks the item number passed to it. For the Stop Printing button, MyDoHandleHitsInStatusDBox calls PrSetError, specifying the error code iPrAbort. For all other items, MyDoHandleHitsInStatusDBox sets the cursor to a spinning wristwatch cursor.

Finally, the MyDoPrintIdle procedure updates the items in the status dialog box that report status to the user.

Handling Printing Errors

You should always check for error conditions while printing by calling the PrError function. Errors returned may include AppleTalk and Operating System errors in addition to Printing Manager errors.

Note

Don't call PrError from within your idle procedure. See "Writing an Idle Procedure" on page 9-38 for more information. ◆

If you determine that an error has occurred after the completion of a printing routine, stop printing. Call the close routine that matches any open routine you have called. For example, if you call PrOpenDoc and receive an error, skip to the next call to PrCloseDoc; if you call PrOpenPage and get an error, skip to the next calls to PrClosePage and PrCloseDoc. Remember that, if you have called some open routine, you must call the corresponding close routine to ensure that the printer driver closes properly and that all temporary memory allocations are released and returned to the heap.

If you are using the `PrError` function and the `PrGeneral` procedure (described in "Getting and Setting Printer Information" beginning on page 9-28), be prepared to receive the following errors: `noSuchRsl`, `opNotImpl`, and `resNotFound`. In all three cases, your application should be prepared to continue to print without using the features of that particular opcode.

The `noSuchRsl` error means that the currently selected printer does not support the requested resolution. The `opNotImpl` error means that the currently selected printer does not support the particular `PrGeneral` opcode that you selected. The `resNotFound` error means the current printer driver does not support the `PrGeneral` procedure at all. This lack of support should not be a problem for your application, but you need to be prepared to deal with this error. If you receive a `resNotFound` result code from `PrError`, clear the error with a call to `PrSetError` with a value of `noErr` as the parameter; otherwise, `PrError` might still contain this error the next time you check it, which would prevent your application from printing.

Do not display any alert or dialog boxes to report an error until the end of the printing loop. Once at the end, check for the error again; if there is no error, assume that the printing completed normally. If the error is still present, then you can alert the user. This technique is important for two reasons.

- First, if you display a dialog box in the middle of the printing loop, it could cause errors that might terminate an otherwise normal printing operation. For example, if the printer is connected to an AppleTalk network, the connection might be terminated abnormally because the printer driver would be unable to respond to AppleTalk requests received from the printer while the dialog box was waiting for input from the user. If the printer does not hear from the Macintosh Operating System within a short period of time (anywhere from 30 seconds to 2 minutes, depending on the driver), it assumes that the Macintosh computer is no longer connected to the printer and times out. The timeout results in a prematurely broken connection, causing another error, to which the application must respond.
- Second, the printer driver may have already displayed its own dialog box in response to an error. In this instance, the printer driver posts an error to let the application know that something went wrong and it should cancel printing. For example, when a LaserWriter printer driver detects that the user has canceled printing, the driver posts an error to let the application know that it needs to cancel printing. Because the driver has already taken care of the error by displaying a dialog box, the error is reset to 0 before the printing loop is complete. The application should check for the error again at the end of the printing loop, and, if appropriate, the application can then display a dialog box.

Printing Manager Reference

This section describes the data structures and routines defined by the Printing Manager. When you print a document using the Printing Manager, the Printing Manager uses a printer driver to do the actual printing. A printer driver does any necessary translation of QuickDraw drawing routines and—when requested by your application—sends the translated instructions and data to the printer. It is the printer driver for the current printer that actually implements the routines defined by the Printing Manager. Every Printing Manager routine you call determines the current printer from a resource in the System file and then dispatches your call to the printer driver for that printer.

"Data Structures" shows the Pascal data structures defined by the Printing Manager. "Printing Manager Routines" describes the routines you can use to open and close the Printing Manager, display a print dialog box, print a document, and handle printing errors. "Application-Defined Routines" describes how you can provide your own idle procedure that handles events in a dialog box reporting the status of the print job, and how you can provide an initialization function that appends items to a print dialog box.

IMPORTANT

The burden of maintaining backward compatibility with early Apple printer models—as well as maintaining compatibility with over a hundred existing printer drivers—requires extra care on your part. When the Printing Manager was initially designed, it was intended to support ImageWriter printers directly attached to Macintosh computers with only a single floppy disk and 128 KB of RAM. Later, the Printing Manager was implemented on PostScript LaserWriter printer drivers for more powerful Macintosh computers sharing LaserWriter printers on networks. Since then, the Printing Manager has been implemented on a substantial—and unanticipated—number of additional Apple and third-party printer drivers, each in its own, slightly unique way. When you use Printing Manager routines and data structures, you should be especially wary of and defensive about possible error conditions. Because Apple has little control over the manner in which third parties support the Printing Manager in their printer drivers, you should test your application's printing code on as many printers as possible. ▲

Data Structures

This section shows the Pascal data structures defined by the Printing Manager.

You must create or ensure a valid `TPrint` record for every document before you can print it. This record specifies printer characteristics and the characteristics of a particular print job.

Contained in every `TPrint` record is a `TPrInfo` record, which specifies the vertical and horizontal resolutions, of the current printer and describes the page rectangle. A `TPrJob` record, which contains information about a particular print job—such as the range of pages to print, the number of copies, and a pointer to an idle procedure—is also contained in a `TPrint` record. A `TPrStl` record, which is also contained in a `TPrint` record, contains the device number of the current printer and the feed type to be used when printing the document.

The `PrPicFile` procedure returns printing status information in a record of data type `TPrStatus`. (You call the `PrPicFile` procedure for a printer using deferred printing.)

The `TPrDlg` record contains information necessary when altering the default style or job dialog box.

The `TPrPort` record describes a printing graphics port—the environment into which your application draws in order to print.

You use the `TGnlData`, `TGetRslBlk`, `TSetRslBlk`, `TDftBitsBlk`, and `TGetRotnBlk` records in conjunction with the `PrGeneral` procedure.

In almost all cases, printer drivers use the reserved fields in these data structures for device-dependent information. You should not rely on the availability or accuracy of this information when printing from your application.

TPrint

You must supply a record of data type `TPrint` for a document before it can be printed. (See “Creating and Using a TPrint Record” beginning on page 9-17 for information about how to supply a `TPrint` record for a document.)

In addition to other fields, the `TPrint` record includes three fields (`prInfo`, `prStl`, and `prJob`) that are defined by the `TPrInfo` record (described on page 9-46), the `TPrStl` record (described on page 9-48), and the `TPrJob` record (described on page 9-47). The `TPrint` record and the records within it contain information such as that needed by your application for printing a document.

```
TYPE
   TPPrint = ^TPrint;      {pointer to a TPrint record}
   THPrint = ^TPPrint;     {handle to a TPrint record}
   TPrint =
   RECORD
      iPrVersion: Integer; {reserved}
      prInfo:     TPrInfo; {resolution of device & page rectangle}
      rPaper:     Rect;    {paper rectangle}
      prStl:      TPrStl;  {printer driver number & feed type}
      prInfoPT:   TPrInfo; {reserved}
      prXInfo:    TPrXInfo;{reserved}
      prJob:      TPrJob;  {printing information from the job }
                           { dialog box}
      printX:     ARRAY[1..19] OF Integer;
   END;
```

Field descriptions

`iPrVersion` Reserved. To determine the version of the printer driver that initialized this `TPrint` record, use the `PrDrvrVers` function, which is described on page 9-79.

`prInfo` The information needed for page composition, contained in a `TPrInfo` record. See page 9-46 for a description of this record.

`rPaper` The paper rectangle. This rectangle encompasses the page rectangle, which is specified by the `rPage` field of the `TPrInfo` record.

`prStl` The printer's device number and the feed type, contained in a `TPrStl` record. See page 9-48 for a description of this record.

`prInfoPT` Reserved.

`prXInfo` Reserved.

`prJob` Information about this particular print job, contained in a `TPrJob` record. You use the `PrJobDialog` function to display the job dialog box. After the user closes the job dialog box, the `PrJobDialog` function updates the fields of the `TPrJob` record according to the user's choices. See page 9-47 for a description of this record.

`printX` Reserved.

If you try to use a `TPrint` record that's invalid for the current version of the Printing Manager or for the current printer, the printer driver corrects the record by setting its fields to default values.

Your application should not directly change the user-supplied data in the `TPrint` record; your application should use the `PrStlDialog` function and the `PrJobDialog` function (described on page 9-61 and page 9-62, respectively) or the `PrDlgMain` function (described on page 9-63) to allow the user to specify printing options, which the printer driver then translates to the appropriate fields in the `TPrint` record. The only fields you may need to set directly are those containing optional information in the `TPrJob` record (for example, the `pIdleProc` field, which contains a pointer to an idle procedure). Attempting to set other values directly in the `TPrint` record can produce unexpected results.

TPrInfo

The record defined by the data type `TPrInfo` contains printer information. The `prInfo` field of the `TPrint` record (described in the preceding section) contains a `TPrInfo` record, which in turn contains the vertical and horizontal resolutions of the printer and the coordinates of the page rectangle.

```
TYPE TPrInfo =          {printer information record}
   RECORD
      iDev:    Integer; {reserved}
      iVRes:   Integer; {vertical resolution of printer, in dpi}
      iHRes:   Integer; {horizontal resolution of printer, in dpi}
      rPage:   Rect;    {the page rectangle}
   END;
```

Field descriptions

`iDev` Reserved.

`iVRes` The printer's vertical resolution in dots per inch. The default value is 72, unless you have previously set the value for this record by calling the `PrGeneral` procedure with the `setRslOp` opcode (described in "Determining and Setting the Resolution of the Current Printer" on page 9-30).

`iHRes` The printer's horizontal resolution in dots per inch. The default value is 72, unless you have previously set the value for this record by calling the `PrGeneral` procedure with the `setRslOp` opcode.

`rPage` The page rectangle. As illustrated in Figure 9-6 on page 9-10, this rectangle is inside the paper rectangle, which is specified by the `rPaper` field of the `TPrint` record, described on page 9-44. You use the `PrStlDialog` function (described on page 9-61) to display the style dialog box. After the user closes the style dialog box, the `PrStlDialog` function updates the `rPage` field according to the user's choices.

TPrJob

The record defined by the data type `TPrJob` contains information about the print job. The `prJob` field of the `TPrint` record (described on page 9-44) contains a `TPrJob` record. You can set the contents of this record as a result of calling the `PrJobDialog` function (described on page 9-62) or the `PrJobInit` function (described on page 9-65), or by calling the `PrintDefault` procedure or `PrValidate` function (described on page 9-59 and page 9-60, respectively).

```
TYPE TPrJob =                  {print job record}
   RECORD
      iFstPage:   Integer;     {first page of page range}
      iLstPage:   Integer;     {last page of page range}
      iCopies:    Integer;     {number of copies}
      bJDocLoop:  SignedByte; {printing method: draft or deferred}
      fFromUsr:   Boolean;     {reserved}
      pIdleProc:  PrIdleProcPtr;
                               {pointer to an idle procedure}
      pFileName:  StringPtr;   {spool filename: NIL for default}
      iFileVol:   Integer;     {spool file volume; set to 0 }
                               { initially}
      bFileVers:  SignedByte; {spool file version; set to 0 }
                               { initially}
      bJobX:      SignedByte; {reserved}
END;
```

Field descriptions

`iFstPage` The page number of the first page to print.

`iLstPage` The page number of the last page to print.

`iCopies` The number of copies requested, which is also the number of times your application should send the document to the printer. However, some PostScript printer drivers handle multiple copies internally and set this value to 1.

`bJDocLoop` The printing method, as indicated by one of these constants:

```
CONST bDraftLoop = 0;   {draft-quality printing}
      bSpoolLoop = 1;   {deferred printing}
```

See the description of the `PrPicFile` procedure on page 9-71 on how to send a print job to the printer when this field contains the `bSpoolLoop` constant.

`fFromUsr` Reserved.

pIdleProc	A pointer to the idle procedure (described in "Writing an Idle Procedure" on page 9-38) for this printing operation. A value of NIL specifies the default idle procedure.
pFileName	The name of the spool file (normally "Print File") for deferred printing. This field is maintained by the printer driver, and your application should not change or rely on its value.
iFileVol	The volume reference number of the spool file. This field is maintained by the printer driver, and your application should not change or rely on its value.
bFileVers	The version number of the spool file, initialized to 0.
bJobX	Reserved.

TPrStl

The prStl field of the TPrint record (described on page 9-44) contains a TPrStl record, which in turn contains the device number of the current printer and the feed type currently selected (either paper cassette or manual). All other fields are reserved.

```
TYPE TPrStl =              {printing style record}
   RECORD
      wDev:    Integer;    {device number of printer}
      iPageV:  Integer;    {reserved}
      iPageH:  Integer;    {reserved}
      bPort:   SignedByte; {reserved}
      feed:    TFeed;      {feed type}
   END;
```

Field descriptions

wDev	The device number of the current printer (in the high-order byte of this field). The low-order byte of this field is reserved.
iPageV	Reserved.
iPageH	Reserved.
bPort	Reserved.
feed	The feed type currently selected. The possible values are defined by the TFeed data type:

```
TYPE TFeed =
(feedCut,feedFanfold,feedMechCut,feedOther);
```

TPrStatus

The PrPicFile procedure (described on page 9-71) returns printing status information in a record of data type TPrStatus. (You call the PrPicFile procedure for a printer using deferred printing.)

```
TYPE TPrStatus =              {printing status record}
   RECORD
      iTotPages:  Integer;    {total pages in print file}
      iCurPage:   Integer;    {current page number}
      iTotCopies: Integer;    {total copies requested}
      iCurCopy:   Integer;    {current copy number}
      iTotBands:  Integer;    {reserved}
      iCurBand:   Integer;    {reserved}
      fPgDirty:   Boolean;    {TRUE if current page has been }
                              { written to}
      fImaging:   Boolean;    {reserved}
      hPrint:     THPrint;    {handle to the active TPrint record}
      pPrPort:    TPPrPort;   {pointer to the active printing }
                              { port}
      hPic:       PicHandle;  {handle to the active picture}
   END;
```

Field descriptions

iTotPages	The total number of pages being printed. This is the value of the iLstPage field minus the value of the iFstPage field, which are both in the TPrJob record (described on page 9-47).
iCurPage	The sequence number of the page currently being printed. For example, if the user prints pages 10 through 15 of a 20-page document, the value of the iCurPage field for page 10 is 1.
iTotCopies	The total number of copies requested. This value may be different from the value of the iCopies field in the TPrJob record.
iCurCopy	The number of the current copy being printed.
iTotBands	Reserved.
iCurBand	Reserved.
fPgDirty	A flag indicating whether the printer has begun printing the current page. Set to TRUE if there has been any imaging on the current page.
fImaging	A flag indicating whether the printer driver is in the middle of an imaging call.
hPrint	A handle to the current TPrint record (described on page 9-44).
pPrPort	A pointer to the TPrPort record for the current printing graphics port (described on page 9-51).
hPic	A handle to the active picture. This is used by the printer driver; your application should not alter it.

TPrDlg

The `TPrDlg` record contains information necessary when altering the default style or job dialog box for the current printer driver. The `PrStlInit` function (described on page 9-64) returns a `TPrDlg` record with information for a style dialog box; the `PrJobInit` function (described on page 9-65) returns a `TPrDlg` record with information for a job dialog box.

```
TYPE
TPPrDlg  = TPrDlg;
TPrDlg   =                          {print dialog box record}
   RECORD
      Dlg:        DialogRecord;     {a dialog record}
      pFltrProc:                    {pointer to event filter}
                  ModalFilterProcPtr;
      pItemProc:  PItemProcPtr;     {pointer to item-handling }
                                    { procedure}
      hPrintUsr:  THPrint;          {handle to a TPrint record}
      fDoIt:      Boolean;          {TRUE means user clicked OK}
      fDone:      Boolean;          {TRUE means user clicked }
                                    { OK or Cancel}
      lUser1:     LongInt;          {storage for your application}
      lUser2:     LongInt;          {storage for your application}
      lUser3:     LongInt;          {storage for your application}
      lUser4:     LongInt;          {storage for your application}
   END;
```

Field descriptions

`Dlg` A dialog record that represents either the style or job dialog box. This record is described in the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

`pFltrProc` A pointer to an event filter function that handles events the Dialog Manager does not respond to (such as disk-inserted events and update events for background applications) in a modal dialog box. Event filter functions for modal dialog boxes are described in the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

`pItemProc` A pointer to a routine (sometimes called a *dialog hook*) that responds to events in those items—such as checkboxes and radio buttons—that your application has added to the dialog box. See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about responding to events in dialog boxes.

`hPrintUsr`	A handle to a `TPrint` record (described on page 9-44) for a document.
`fDoIt`	A Boolean value indicating whether the user has confirmed the dialog box. A value of `TRUE` means the user has confirmed it by clicking the OK button.
`fDone`	A Boolean value indicating whether the user's interaction is completed. A value of `TRUE` means the user has clicked either the OK or Cancel button.
`lUser1`	In this field and the following fields, your application can store any kind of data you wish for the dialog box.
`lUser2`	Available for your application's use.
`lUser3`	Available for your application's use.
`lUser4`	Available for your application's use.

Figure 9-3 on page 9-7 shows a style dialog box. Figure 9-5 on page 9-8 shows a job dialog box. You can find information on how to customize a dialog box in "Altering the Style or Job Dialog Box" beginning on page 9-35.

TPrPort

The record of the data type `TPrPort` contains a graphics port and a pointer to a `QDProcs` record.

```
TYPE TPrPort =                {printing graphics port}
   RECORD
      gPort:      GrafPort;   {graphics port for printing}
      gProcs:     QDProcs;    {procedures for printing }
                              { in the graphics port}
      lGParam1:   LongInt;    {reserved}
      lGParam2:   LongInt;    {reserved}
      lGParam3:   LongInt;    {reserved}
      lGParam4:   LongInt;    {reserved}
      fOurPtr:    Boolean;    {reserved}
      fOurBits:   Boolean;    {reserved}
END;
```

Field descriptions

`gPort`	Either a `CGrafPort` or `GrafPort` record, depending on whether the current printer supports color or grayscale and depending on whether Color QuickDraw is available. If you need to determine the type of graphics port, you can check the high bit in the `rowBytes` field of the record contained in the `gPort` field; if this bit is set, the printing graphics port is based on a `CGrafPort` record.

gProcs	A QDProcs record that contains pointers to routines that the printer driver may have designated to take the place of QuickDraw routines. See the chapter "Basic QuickDraw" in this book for more information about the QDProcs record.
lGParam1	Reserved.
lGParam2	Reserved.
lGParam3	Reserved.
lGParam4	Reserved.
fOurPtr	Reserved.
fOurBits	Reserved.

TGnlData

The record of data type TGnlData is the basic record used by the PrGeneral procedure (described beginning on page 9-72). Although no opcode of PrGeneral uses the TGnlData record, all other records created for PrGeneral are based on this record.

```
TYPE TGnlData =
   RECORD
      iOpCode:    Integer; {opcode passed to PrGeneral}
      iError:     Integer; {result code returned by PrGeneral}
      lReserved:  LongInt; {more fields here depending on opcode}
END;
```

Field descriptions

iOpCode — The opcode that is passed to PrGeneral to obtain the requested feature. There are five possible opcodes; you can use the following constants or the values they represent to specify one of these opcodes

```
CONST {opcodes used with PrGeneral}
 getRslDataOp  = 4;  {get resolutions for the }
                     { current printer}
 setRslOp      = 5;  {set resolutions for a }
                     { TPrint record}
 draftBitsOp   = 6;  {force enhanced draft- }
                     { quality printing}
 noDraftBitsOp = 7;  {cancel enhanced draft- }
                     { quality printing}
 getRotnOp     = 8;  {get page orientation of }
                     { a TPrint record}
```

iError — The result code returned by PrGeneral.

lReserved — Reserved. Additional fields may follow this field, depending on the opcode used. See the descriptions of the TGetRslBlk (in the next section), TSetRslBlk (on page 9-54), TDftBitsBlk (on page 9-55), and TGetRotnBlk (on page 9-56) records.

TGetRslBlk

You pass a record defined by the data type TGetRslBlk to the PrGeneral procedure when you use the getRslDataOp opcode. When the PrGeneral procedure completes, the TGetRslBlk record contains the resolutions available on the current printing device. For information on how to use the TGetRslBlk record with the PrGeneral procedure, see "Determining and Setting the Resolution of the Current Printer" on page 9-30.

```
TYPE TGetRslBlk =          {get-resolution record}
   RECORD
      iOpCode:    Integer; {the getRslDataOp opcode}
      iError:     Integer; {result code returned by PrGeneral}
      lReserved:  LongInt; {reserved}
      iRgType:    Integer; {printer driver version number}
      xRslRg:     TRslRg;  {x-direction resolution range}
      yRslRg:     TRslRg;  {y-direction resolution range}
      iRslRecCnt: Integer; {number of resolution records}
      rgRslRec:            {array of resolution records}
                  ARRAY[1..27] OF TRslRec;
   END;
```

Field descriptions

iOpCode — The opcode getRslDataOp.

iError — The result code returned by PrGeneral.

lReserved — Reserved.

iRgType — The version number returned by the printer driver.

xRslRg — The resolution range supported for the x direction. This field contains a record defined by the data type TRslRg:

```
TYPE TRslRg =
   RECORD
      iMin: Integer; {minimum resolution supported}
      iMax: Integer; {maximum resolution supported}
   END;
```

If the current printer does not support variable resolution, the values in the iMin and iMax fields are 0.

yRslRg — The resolution range supported for the y direction. This field contains a record defined by the data type TRslRg, which is shown in the preceding description for the xRslRg field. If the current printer does not support variable resolution, the values in the iMin and iMax fields are 0.

iRslRecCnt — The number of TRslRec records used by a particular printer driver (up to 27) if it supports discrete resolution. If it supports variable resolution, this field contains 0. The TRslRec record is described next.

rgRslRec — An array of records defined by the TRslRec data type, each specifying a discrete resolution at which the current printer can print an image. A printer driver may contain up to 27 separate TRslRec records.

```
TYPE TRslRec =
   RECORD
      iXRsl: Integer;   {discrete resolution, }
                        { x direction}
      iYRsl: Integer;   {discrete resolution, }
                        { y direction}
   END;
```

TSetRslBlk

You pass a record defined by the data type TSetRslBlk to the PrGeneral procedure when you use the setRslOp opcode. You use this record to specify the resolutions that you want to use when printing the data associated with a TPrint record. For information on how to use the TSetRslBlk record with the PrGeneral procedure, see "Determining and Setting the Resolution of the Current Printer" beginning on page 9-30.

```
TYPE TSetRslBlk =          {set-resolution record}
   RECORD
      iOpCode:    Integer; {the setRslOp opcode}
      iError:     Integer; {result code returned by PrGeneral}
      lReserved:  LongInt; {reserved}
      hPrint:     THPrint; {handle to the current TPrint record}
      iXRsl:      Integer; {x-direction resolution you want}
      iYRsl:      Integer; {y-direction resolution you want}
   END;
```

Field descriptions

iOpCode — The opcode setRslOp.

iError — The result code returned by PrGeneral.

lReserved — Reserved.

hPrint	A handle to a TPrint record, which is described on page 9-44. Your application should have already created this TPrint record and passed it to the PrintDefault or PrValidate routine to make sure that all of the information in the TPrint record is valid.
iXRsl	The resolution in the x direction that you want the printer to use when printing the data associated with the TPrint record specified in the hPrint field.
iYRsl	The resolution in the y direction that you want the printer to use when printing the data associated with the TPrint record specified in the hPrint field.

After calling PrGeneral with the setRslOp opcode, you can determine whether the request was successful by examining the iError field of the TSetRslBlk record. If the iError field returns noErr, the Printing Manager updated the TPrint record with the specified resolution, which the printer uses when printing the data associated with this TPrint record. If the iError field returns noSuchRsl, the current printer doesn't support the requested resolution, and the printer driver does not change the setting in the TPrint record.

TDftBitsBlk

You pass a record defined by the data type TDftBitsBlk to the PrGeneral procedure when you use the draftBitsOp or noDraftBitsOp opcode. For information on how to use the TDftBitsBlk record with the PrGeneral procedure, see "Enhancing Draft-Quality Printing" on page 9-33.

```
TYPE TDftBitsBlk =         {draft bits record}
   RECORD
      iOpCode:    Integer; {draftBitsOp or noDraftBitsOp opcode}
      iError:     Integer; {result code returned by PrGeneral}
      lReserved:  LongInt; {reserved}
      hPrint:     THPrint; {handle to the current TPrint record}
   END;
```

Field descriptions

iOpCode	Either the draftBitsOp or noDraftBitsOp opcode.
iError	The result code returned by the PrGeneral procedure.
lReserved	Reserved.
hPrint	A handle to a TPrint record, which is described on page 9-44. Your application should have already created this TPrint record and passed it to the PrintDefault or PrValidate routine to make sure that all of the information in the TPrint record is valid. The PrintDefault and PrValidate routines are described on page 9-59 and page 9-60, respectively.

TGetRotnBlk

You pass a record defined by the data type `TGetRotnBlk` to the `PrGeneral` procedure when you use the `getRotnOp` opcode. `PrGeneral` returns it with a Boolean variable that tells you whether the user has selected landscape orientation. For information on how to use the `TGetRotnBlk` record with the `PrGeneral` procedure, see "Determining Page Orientation" on page 9-32.

```
TYPE TGetRotnBlk =            {page orientation record}
   RECORD
      iOpCode:    Integer;    {the getRotnOp opcode}
      iError:     Integer;    {result code returned by PrGeneral}
      lReserved:  LongInt;    {reserved}
      hPrint:     THPrint;    {handle to current TPrint record}
      fLandscape: Boolean;    {TRUE if user selected landscape }
                              { printing}
      bXtra:      SignedByte; {reserved}
   END;
```

Field descriptions

`iOpCode` The opcode `getRotnOp`.

`iError` The result code returned by the `PrGeneral` procedure.

`lReserved` Reserved.

`hPrint` A handle to a `TPrint` record, which is described on page 9-44. Your application should have already created this `TPrint` record and passed it through the `PrintDefault` or `PrValidate` routine to make sure that all of the information in the `TPrint` record is valid. The `PrintDefault` and `PrValidate` routines are described on page 9-59 and page 9-60, respectively.

`fLandscape` A Boolean value that determines whether the user has selected landscape orientation in the style dialog box. A value of `TRUE` indicates the user has selected landscape orientation.

`bXtra` Reserved.

Printing Manager Routines

This section describes the routines you use to open and close the current printer driver, produce or alter a style or job dialog box, print a document, and handle printing errors.

Opening and Closing the Printing Manager

You must always use the `PrOpen` procedure to open the current printer driver before attempting to print, and you must use the `PrClose` procedure to close the current printer driver when printing is finished.

PrOpen

Use the `PrOpen` procedure to prepare the current printer driver for use.

```
PROCEDURE PrOpen;
```

DESCRIPTION

The `PrOpen` procedure opens the Printing Manager and the current printer driver.

SPECIAL CONSIDERATIONS

You must always use the `PrOpen` procedure before using any other Printing Manager routines, and you must balance every call to `PrOpen` with a call to `PrClose`, which is described in the next section.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrOpen` procedure are

Trap macro	Selector
`_PrGlue`	$C8000000

SEE ALSO

For an example of the use of `PrOpen`, see Listing 9-2 on page 9-20.

PrClose

When you are finished using Printing Manager routines, use the `PrClose` procedure to close the Printing Manager and release the memory it occupies.

```
PROCEDURE PrClose;
```

DESCRIPTION

The `PrClose` procedure is the call that balances a call to the `PrOpen` procedure.

SPECIAL CONSIDERATIONS

If you have opened the printer driver with the `PrOpen` procedure, do not call the `PrDrvrClose` procedure (described on page 9-80) to close it. Similarly, do not close the printer driver with `PrClose` if you opened it with the `PrDrvrOpen` procedure (described on page 9-79).

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrClose` procedure are

Trap macro	Selector
`_PrGlue`	$D0000000

SEE ALSO

For an example of the use of `PrClose`, see Listing 9-2 beginning on page 9-20.

Initializing and Validating TPrint Records

You must set the fields of the `TPrint` record to the values for the current printer driver or, if a `TPrint` record already exists, you must verify that the information in the `TPrint` record is correct. The `PrintDefault` procedure fills in a `TPrint` record with the default values for the current printer.

If the `TPrint` record is not valid for the current printer driver, the document does not print. The `PrValidate` function ensures that the `TPrint` record is compatible with the current version of the printer driver for the current printer. These functions may change the coordinates of the page rectangle or any other value in the `TPrint` record; you should not assume any values will remain the same.

PrintDefault

When you create a `TPrint` record, you use the `PrintDefault` procedure to initialize the fields of the `TPrint` record according to the current printer's default values for resolution, number of copies, and so on.

```
PROCEDURE PrintDefault (hPrint: THPrint);
```

`hPrint` A handle to a `TPrint` record (described on page 9-44), which may be a new record or an existing one from a document.

DESCRIPTION

The default values for the current printer are stored in the printer driver's resource file. The `PrintDefault` procedure puts these values in the `TPrint` record, replacing the ones that may already be there. The `PrintDefault` procedure calls the `PrValidate` function (described in the next section) to ensure that the `TPrint` record is compatible with the current version of the printer driver.

SPECIAL CONSIDERATIONS

You should never call `PrintDefault` between the pages of a document.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrintDefault` procedure are

Trap macro	Selector
`_PrGlue`	$20040480

SEE ALSO

See "The TPrint Record and the Printing Loop" on page 9-11 and see page 9-44 for information on the `TPrint` record. For an example of the use of `PrintDefault`, see Listing 9-7 on page 9-37.

PrValidate

When you have a `TPrint` record, whether an existing one from the current document or a new one you have just created, you can use the `PrValidate` function to ensure that the contents of the specified `TPrint` record are compatible with the current version of the printer driver for the current printer.

```
FUNCTION PrValidate (hPrint: THPrint): Boolean;
```

`hPrint` A handle to a `TPrint` record, which may be a new record or an existing one from a document.

DESCRIPTION

If the `TPrint` record is valid, the `PrValidate` function returns `FALSE`, meaning there is no change. If the record is invalid, the function returns `TRUE` and the Printing Manager adjusts the record with the default values stored in the printer resource file for the current printer.

The `PrValidate` function also makes sure that all the information in the `TPrint` record is internally self-consistent and updates the `TPrint` record as necessary. These changes do not affect the function's Boolean result.

If you have just created a `TPrint` record by using the `PrintDefault` procedure, you do not need to call `PrValidate`. The `PrintDefault` procedure does this automatically.

SPECIAL CONSIDERATIONS

You should never call `PrValidate` between the pages of a document. This restriction holds as well for the `PrStlDialog` and `PrJobDialog` functions (described on page 9-61 and page 9-62, respectively) and the `PrintDefault` procedure (described on page 9-59), which call `PrValidate`.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrValidate` function are

Trap macro	Selector
`_PrGlue`	$52040498

SEE ALSO

For examples of the use of `PrValidate`, see Listing 9-1 on page 9-17 and Listing 9-2 on page 9-20.

Displaying and Customizing the Print Dialog Boxes

The style and job dialog boxes allow the user to tell your application how to print the document: its page orientation, number of copies, page range, and so on. The `PrStlDialog` and `PrJobDialog` functions display the standard style and job dialog boxes as provided by the resource file of the current printer driver. The `PrDlgMain` function, along with the `PrStlInit` and `PrJobInit` functions, allows you to customize the current printer driver's style and job dialog boxes.

The `PrJobMerge` procedure allows you to use one job dialog box for several print jobs, such as when the user prints several documents from the Finder.

PrStlDialog

You can use the `PrStlDialog` function to display the style dialog box provided by the resource file for the current printer driver.

```
FUNCTION PrStlDialog (hPrint: THPrint): Boolean;
```

`hPrint` A handle to a `TPrint` record (described on page 9-44), which may be a new record or an existing one from a document.

DESCRIPTION

The `PrStlDialog` function gets the initial settings to display in the style dialog box from the `TPrint` record specified in the `hPrint` parameter. The user specifies the page dimensions and other information needed for page setup through the style dialog box. Your application should display this dialog box when the user chooses Page Setup from the File menu.

If the user confirms the dialog box, the `PrStlDialog` function returns `TRUE`. The `PrStlDialog` function saves the results of the dialog box in the specified `TPrint` record and calls the `PrValidate` function (described on page 9-60). Otherwise, the `TPrint` record is left unchanged and the function returns `FALSE`.

SPECIAL CONSIDERATIONS

You should never call `PrStlDialog` between the pages of a document.

You must call the `PrOpen` procedure (described on page 9-57) prior to calling `PrStlDialog`, and you must call the `PrClose` procedure (described on page 9-58) afterward, because the current printer driver must be open in order for your application to successfully call `PrStlDialog`.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrStlDialog` function are

Trap macro	Selector
`_PrGlue`	$2A040484

SEE ALSO

See Figure 9-3 on page 9-7 for an example of a style dialog box. For more information on the use of a style dialog box, see "Getting Printing Preferences From the User" beginning on page 9-5. For information on how to customize a style dialog box, see "Altering the Style or Job Dialog Box" beginning on page 9-35.

PrJobDialog

You can use the `PrJobDialog` function to display the job dialog box provided by the resource file for the current printer driver.

```
FUNCTION PrJobDialog (hPrint: THPrint): Boolean;
```

`hPrint` A handle to a `TPrint` record (described on page 9-44), which may be a new record or an existing one from a document.

DESCRIPTION

The `PrJobDialog` function gets the initial settings to display in the job dialog box from the `TPrint` record specified in the `hPrint` parameter. The user specifies the print quality, the range of pages to print, and other information in the job dialog box. Your application should display this dialog box when the user chooses Print from the File menu.

If the user confirms the dialog box, the `PrJobDialog` function updates both the `TPrint` record and the printer driver resource file and calls the `PrValidate` function, and the `PrJobDialog` function returns `TRUE`. Even if the function returns `FALSE`, the `PrJobDialog` function may have updated the `TPrint` record.

SPECIAL CONSIDERATIONS

You should proceed with the requested printing operation only if the `PrJobDialog` function returns `TRUE`. You should never call `PrJobDialog` between the pages of a document.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrJobDialog` function are

Trap macro	Selector
`_PrGlue`	$32040488

SEE ALSO

See Figure 9-5 on page 9-8 for an example of a job dialog box. For more information on the use of a job dialog box, see "Getting Printing Preferences From the User" beginning on page 9-5. For information on how to customize a job dialog box, see "Altering the Style or Job Dialog Box" beginning on page 9-35.

PrDlgMain

To display a customized style or job dialog box for the current printer driver, use the `PrDlgMain` function.

```
FUNCTION PrDlgMain (hPrint: THPrint; pDlgInit: PDlgInitProcPtr):
                    Boolean;
```

`hPrint` A handle to a `TPrint` record (described on page 9-44), which may be a new record or an existing one from a document.

`pDlgInit` A pointer to your own initialization procedure or a pointer to one of the default initialization functions (`PrStlInit`, which is described in the next section, or `PrJobInit`, which is described on page 9-65).

DESCRIPTION

You use the `PrDlgMain` function to display a style or job dialog box that your application has altered. (If you use the standard style and job dialog boxes, you do not need to call `PrDlgMain`; instead, you can simply call the `PrStlDialog` or `PrJobDialog` function, described on page 9-61 and page 9-62, respectively.)

If you want to customize a style or job dialog box, first call `PrStlInit`, which is described in the next section, or `PrJobInit`, which is described on page 9-65, to get a pointer to the `TPrDlg` record (described on page 9-50) for that dialog box. The `PrStlInit` function returns a pointer to the `TPrDlg` record for the style dialog box of the current printer driver; the `PrJobInit` function returns a pointer to the `TPrDlg` record of the job dialog box for the current printer driver. You should supply the `TPrDlg` record for your customized dialog box with a function that handles events that the Dialog Manager doesn't handle, and with another function that handles events in the items you add to the dialog box.

When `PrDlgMain` returns `TRUE`, you should proceed with the requested printing operation.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrDlgMain` function are

Trap macro	Selector
`_PrGlue`	$4A040894

SEE ALSO

For more information about customizing style or job dialog boxes, see "Altering the Style or Job Dialog Box" beginning on page 9-35.

PrStlInit

To initialize a `TPrDlg` record for a customized style dialog box, use the `PrStlInit` function.

```
FUNCTION PrStlInit (hPrint: THPrint): TPPrDlg;
```

`hPrint` A handle to a `TPrint` record (described on page 9-44), which may be a new record or an existing one from a document.

DESCRIPTION

The `PrStlInit` function returns a pointer to a `TPrDlg` record (described on page 9-50) for the style dialog box defined in the resource file for the current printer driver. As described in "Altering the Style or Job Dialog Box" beginning on page 9-35, you can then alter the dialog box by adding your own items. You must use the `PrDlgMain` function (described on page 9-63) to display the dialog box.

You need to use `PrStlInit` only if you are customizing the default style dialog box provided by the printer driver. To initialize and display the default style dialog box, use the `PrStlDialog` function, which is described on page 9-61.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrStlInit` function are

Trap macro	Selector
`_PrGlue`	$3C04040C

PrJobInit

To initialize a `TPrDlg` record for a customized job dialog box, use the `PrJobInit` function.

```
FUNCTION PrJobInit (hPrint: THPrint): TPPrDlg;
```

`hPrint` A handle to a `TPrint` record (described on page 9-44), which may be a new record or an existing one from a document.

DESCRIPTION

The `PrJobInit` function returns a pointer to a `TPrDlg` record (described on page 9-50) for the job dialog box defined in the resource file for the current printer driver. As described in "Altering the Style or Job Dialog Box" beginning on page 9-35, you can then alter the dialog box by adding your own items. You must use the `PrDlgMain` function (described on page 9-63) to display the dialog box.

You need to use `PrJobInit` only if you are customizing the job dialog box provided by the printer driver. To initialize and display the default job dialog box, use the `PrJobDialog` function, which is described on page 9-62.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrJobInit` function are

Trap macro	Selector
`_PrGlue`	$44040410

SEE ALSO

Listing 9-7 on page 9-37 illustrates how to use `PrJobInit` when customizing the job dialog box.

PrJobMerge

You can use the `PrJobMerge` procedure to apply the same information previously specified by the user through the job dialog box to several `TPrint` records. This is useful when the user prints from the Finder. The `PrJobMerge` procedure allows you to solicit information from the user just once and then use this information to print several documents.

```
PROCEDURE PrJobMerge (hPrintSrc: THPrint; hPrintDst: THPrint);
```

`hPrintSrc` A handle to a `TPrint` record (described on page 9-44) as previously returned by the `PrJobDialog` function (described on page 9-62).

`hPrintDst` A handle to a `TPrint` record for another document.

DESCRIPTION

The `PrJobMerge` procedure first calls the `PrValidate` function (described on page 9-60) for both `TPrint` records referenced by the `hPrintSrc` and `hPrintDst` parameters. It then copies all of the information previously set as a result of a job dialog box from the `TPrint` record in the `hPrintSrc` parameter to the `TPrint` record in the `hPrintDst` parameter while preserving the values set by the style dialog box for that `TPrint` record (for instance, landscape orientation). Finally, the `PrJobMerge` procedure makes sure that all the fields of the `TPrint` record named by the `hPrintDst` parameter are internally self-consistent. You must call `PrJobMerge` for each document the user wants to print.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrJobMerge` procedure are

Trap macro	Selector
`_PrGlue`	$5804089C

Printing a Document

In addition to using the `PrOpen` and `PrClose` procedures (described on page 9-57 and page 9-58, respectively) to open and close the current printer driver, you must open a printing graphics port for a document and open each page of the document before printing the page. You must close each page after printing it, and you must close the printing graphics port after printing the last page of the document. The `PrOpenDoc` function and `PrCloseDoc` procedure open and close the printing graphics port for the document, and the `PrOpenPage` and `PrClosePage` procedures open and close the current page.

You must use the `PrPicFile` procedure to complete printing for a driver using deferred printing.

PrOpenDoc

Use the PrOpenDoc function to initialize a printing graphics port for use in printing a document.

```
FUNCTION PrOpenDoc (hPrint: THPrint; pPrPort: TPPrPort;
                    pIOBuf: Ptr): TPPrPort;
```

hPrint — A handle to a TPrint record (described on page 9-44), which may be a new record or an existing one from a document. You should call the PrintDefault procedure (described on page 9-59) or the PrValidate function (described on page 9-60) for this TPrint record before calling PrOpenDoc.

pPrPort — A pointer to a printing graphics port. If you set this parameter to NIL, PrOpenDoc allocates a new printing graphics port in the heap.

pIOBuf — A pointer to an area of memory to be used as an input and output buffer. If you set this parameter to NIL, PrOpenDoc uses the volume buffer for the deferred spool file's volume. If you allocate your own buffer, it must be exactly 522 bytes.

DESCRIPTION

The PrOpenDoc function initializes and returns a pointer to a printing graphics port for use in printing a document. (The TPrPort record that defines a printing graphics port is described on page 9-51.) The PrOpenDoc function also sets the current graphics port to the printing graphics port.

Because both the printing graphics port and input and output buffer are nonrelocatable objects, you may want to allocate them yourself using the pPrPort and pIOBuf parameters (to avoid fragmenting the heap).

SPECIAL CONSIDERATIONS

You must balance a call to PrOpenDoc with a call to the PrCloseDoc procedure, which is described in the next section.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the PrOpenDoc function are

Trap macro	Selector
_PrGlue	$04000C00

SEE ALSO

For an example of the use of PrOpenDoc, see Listing 9-2 beginning on page 9-20. For a description of the PrValidate function and PrintDefault procedure, see page 9-60 and page 9-59, respectively.

PrCloseDoc

Use the `PrCloseDoc` procedure to close a printing graphics port previously opened with the `PrOpenDoc` procedure.

```
PROCEDURE PrCloseDoc (pPrPort: TPPrPort);
```

`pPrPort` A pointer to a printing graphics port. (The `TPrPort` record that defines a printing graphics port is described on page 9-51.)

DESCRIPTION

The `PrCloseDoc` procedure closes the current printing graphics port. You typically use `PrCloseDoc` after sending the last page of a document to the printer with the `PrClosePage` procedure (described on page 9-70).

When you use `PrCloseDoc` to close a printing graphics port, printer drivers respond in a manner appropriate for the printers they control. Many drivers, including the LaserWriter driver, start a print job after your application calls `PrCloseDoc`.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrCloseDoc` procedure are

Trap macro	Selector
`_PrGlue`	$08000484

SPECIAL CONSIDERATIONS

For deferred printing on an ImageWriter printer, call the `PrError` function (described on page 9-75) to find out whether spooling succeeded before using `PrCloseDoc`. If spooling succeeded, call the `PrPicFile` procedure (described on page 9-71).

SEE ALSO

For an example of the use of `PrCloseDoc`, see Listing 9-2 beginning on page 9-20.

PrOpenPage

Use the `PrOpenPage` procedure to begin to print a new page.

```
PROCEDURE PrOpenPage (pPrPort: TPPrPort; pPageFrame: TPRect);
```

`pPrPort` A pointer to a printing graphics port. (The `TPrPort` record that defines a printing graphics port is described on page 9-51.)

`pPageFrame`
For deferred printing, a pointer to a rectangle to be used as the QuickDraw picture frame for this page. To print the page with no scaling, specify `NIL` to use the rectangle in the `rPage` field of the `TPrInfo` record as the picture frame.

DESCRIPTION

The `PrOpenPage` procedure sets up the printing graphics port to print a new page. After calling `PrOpenPage`, your application should draw the data for that page and then call the `PrClosePage` procedure, which is described in the next section.

The page is printed only if it falls within the page range stored in the `TPrJob` record contained in the `TPrint` record supplied to the `PrOpenDoc` function (described on page 9-67).

If the user has chosen deferred printing for a printer driver that supports deferred printing, the driver uses the QuickDraw procedure `DrawPicture` to scale the rectangle named in the `pPageFrame` parameter so that it coincides with the rectangle specified in the `rPage` field of the `TPrInfo` record (which is contained in the `TPrint` record supplied to the `PrOpenDoc` function). Unless you want the printout to be scaled, you should set the `pPageFrame` parameter to `NIL`—this uses the rectangle in the `rPage` field as the picture frame, so that the page is printed with no scaling.

SPECIAL CONSIDERATIONS

You must balance every call to `PrOpenPage` with a call to `PrClosePage`.

The printing graphics port is completely reinitialized by `PrOpenPage`. Therefore, you must set graphics port features such as the font family and font size for every page that you draw after you call this procedure.

Don't call the QuickDraw function `OpenPicture` while a page is open (after a call to `PrOpenPage` but before calling `PrClosePage`). You can, however, call the `DrawPicture` procedure at any time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrOpenPage` procedure are

Trap macro	Selector
`_PrGlue`	$10000808

SEE ALSO

For an example of the use of `PrOpenPage`, see Listing 9-2 beginning on page 9-20. The QuickDraw routines `OpenPicture` and `DrawPicture` are described in the chapter "Pictures" in this book.

PrClosePage

Use the `PrClosePage` procedure to finish the printing of the current page.

```
PROCEDURE PrClosePage (pPrPort: TPPrPort);
```

`pPrPort` A pointer to a printing graphics port. (The `TPrPort` record that defines a printing graphics port is described on page 9-51.)

DESCRIPTION

The `PrClosePage` procedure records that you are finished printing the current page. The printer driver can then do whatever it requires (such as releasing temporary memory) to avoid communication difficulties or other problems that may cause the user's computer to crash.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrClosePage` procedure are

Trap macro	Selector
`_PrGlue`	$1800040C

SEE ALSO

For an example of the use of `PrClosePage`, see Listing 9-2 beginning on page 9-20.

PrPicFile

Use the `PrPicFile` procedure to complete deferred printing.

```
PROCEDURE PrPicFile (hPrint: THPrint; pPrPort: TPPrPort;
                     pIOBuf: Ptr; pDevBuf: Ptr;
                     VAR prStatus: TPrStatus);
```

`hPrint` A handle to a `TPrint` record (described on page 9-44) for a document.

`pPrPort` A pointer to the printing graphics port. (The `TPrPort` record that defines a printing graphics port is described on page 9-51.) If this parameter is `NIL`, the `PrPicFile` procedure allocates a new printing graphics port in a heap.

`pIOBuf` A pointer to an area of memory to be used as an input/output buffer. This parameter should be `NIL` to use the volume buffer for the spool file's volume. If you allocate your own buffer, it must be exactly 522 bytes.

`pDevBuf` A pointer to a device-dependent buffer. This parameter should be `NIL` so that `PrPicFile` allocates a buffer in a heap.

`prStatus` A `TPrStatus` record that `PrPicFile` uses to report on the current page number, current copy, or current file being spooled. You can then display this information to the user. The `TPrStatus` record is described on page 9-49.

DESCRIPTION

The `PrPicFile` procedure sends a file spooled for deferred printing to the printer.

You can determine whether a user has chosen deferred printing by testing for the `bSpoolLoop` constant in the `bJDocLoop` field of the `TPrJob` record contained in the `TPrint` record specified in the `hPrint` parameter. If the `bJDocLoop` field contains the value represented by the `bSpoolLoop` constant, call the `PrPicFile` procedure, which sends the spool file to the printer.

Your application should normally call `PrPicFile` after the `PrCloseDoc` procedure (described on page 9-68).

SPECIAL CONSIDERATIONS

Do not pass, in the `pPrPort` parameter, a pointer to the same printing graphics port you received from the `PrOpenDoc` function (described on page 9-67). If that port was allocated by `PrOpenDoc` itself (that is, if the `pPrPort` parameter to `PrOpenDoc` was `NIL`), then `PrCloseDoc` will already have disposed of the port, making your pointer to it invalid. Of course, if you earlier provided your own storage in `PrOpenDoc`, there's no reason you can't use the same storage again for `PrPicFile`.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrPicFile` procedure are

Trap macro	Selector
`_PrGlue`	$60051480

SEE ALSO

For an example of the use of `PrPicFile`, see Listing 9-2 beginning on page 9-20.

Optimizing Printing

The `PrGeneral` procedure helps you achieve the highest possible resolution on the current printer, verify page orientation, and force enhanced draft-quality printing for printer drivers supporting these options. To select which action you want, pass one of four records to `PrGeneral` and, in a field of that record, you supply the opcode that specifies the action you require. These records are `TGetRslBlk` (described on page 9-53), `TSetRslBlk` (described on page 9-54), `TGetRotnBlk` (described on page 9-56), and `TDftBitsBlk` (described on page 9-55). All of these records are based on the `TGnlData` record (described on page 9-52), so the first three fields of each are identical.

PrGeneral

Use the `PrGeneral` procedure to achieve the highest possible resolution on the current printer, verify page orientation, and allow enhanced draft-quality printing.

```
PROCEDURE PrGeneral (pData: Ptr);
```

`pData` A pointer to one of these four records, depending on your purpose for calling `PrGeneral`:

A `TGetRslBlk` record (described on page 9-53) for determining resolutions of the current printer. You set the `getRslDataOp` opcode in the `iOpCode` field of this record.

A `TSetRslBlk` record (described on page 9-54) for setting the resolution of a `TPrint` record. In the fields of this record, you specify the `setRslOp` opcode, a handle to a `TPrint` record (described on page 9-44), and the new resolutions for the x and y directions.

A `TGetRotnBlk` record (described on page 9-56) when determining whether to print in landscape orientation. You specify the `getRotnOp` opcode and a handle to a `TPrint` record in the fields of this record.

A `TDftBitsBlk` record (described on page 9-55) to use or cancel enhanced draft-quality printing. You specify in the fields of this record either the `draftBitsOp` or `noDraftBitsOp` opcode and a handle to a `TPrint` record.

DESCRIPTION

To select which action you want the `PrGeneral` procedure to undertake, you pass an opcode in the `iOpCode` field of the record that the `pData` parameter points to.

Use the `PrGeneral` procedure with the value `getRslDataOp` in the `iOpCode` field of a `TGetRslBlk` record when you want to determine the resolutions supported by the current printer driver. The `PrGeneral` procedure returns information about the resolutions that the printer driver supports in the `xRslRg`, `yRslRg`, `iRslRecCnt`, and `rgRslRec` fields of the `TGetRslBlk` record.

Use the `PrGeneral` procedure with the value `setRslOp` in the `iOpCode` field of the `TSetRslBlk` record when you want to set the resolution of a `TPrint` record. When called with the `setRslOp` opcode, `PrGeneral` sets the fields relating to x and y resolution in the specified `TPrint` record according to the values of the `iXRsl` and `iYRsl` fields of the `TSetRslBlk` record.

Use the `PrGeneral` procedure with the value `getRotnOp` in the `iOpCode` field of the `TGetRotnBlk` record when you want to determine whether a `TPrint` record specifies landscape orientation. The `PrGeneral` procedure returns in the `fLandscape` field of this record a Boolean value indicating whether the `TPrint` record specifies landscape orientation. When the user chooses landscape orientation from the style dialog box, the `PrStlDialog` function (described on page 9-61) modifies the `TPrint` record accordingly.

Use the `PrGeneral` procedure with the value `draftBitsOp` in the `iOpCode` field of the `TDftBitsBlk` record when you want to use enhanced draft-quality printing. Typically, you use enhanced draft-quality printing when you want to print bitmaps as well as text in a draft-quality printout on an ImageWriter printer. Use the `noDraftBitsOp` opcode to cancel the use of enhanced draft-quality printing.

If you want to force enhanced draft-quality printing, you should call `PrGeneral` with the `draftBitsOp` opcode before displaying the print dialog boxes to the user. Use of the `draftBitsOp` opcode may cause the printer driver to make some items in its print dialog boxes inactive; for example, the ImageWriter printer driver makes the landscape icon in the style dialog box (landscape printing is not available for draft-quality printing) and the Best and Faster buttons in the job dialog box inactive.

The `PrGeneral` procedure returns error information in the `iError` field of each of these records. You should check the value in the `iError` field after each use of `PrGeneral`. You should also use the `PrError` function (which returns the result code left by the last Printing Manager routine) after checking the `iError` field, to be sure that no additional errors were generated. If `PrError` returns the result code `resNotFound` after you call `PrGeneral`, then the current printer driver doesn't support `PrGeneral`. You should clear the error by calling the `PrSetError` procedure and passing `noErr` in its parameter; otherwise, `PrError` might still contain this error the next time you check it. (The `PrError` function and the `PrSetError` procedure are described on page 9-75 and page 9-78, respectively.)

SPECIAL CONSIDERATIONS

If you call `PrGeneral` with the `draftBitsOp` opcode after using the `PrJobDialog` or `PrDlgMain` function, and if the user chooses draft printing from the job dialog box, the ImageWriter does not print any bitmaps or pixel maps contained in the document.

Enhanced draft-quality printing is of limited usefulness, as described in "Enhancing Draft-Quality Printing" on page 9-33.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrGeneral` procedure are

Trap macro	Selector
`_PrGlue`	$70070480

RESULT CODES

`opNotImpl`	2	Printer driver does not support this opcode
`noSuchRsl`	1	Requested resolution not supported by the currently selected printer
`noErr`	0	No error

SEE ALSO

See Listing 9-4 on page 9-31 for an example of how to use the `getRslDataOp` opcode to determine what printer resolutions are available for the current printer. The same listing shows an example of how to use the `setRslOp` opcode to set the resolution for the current printer.

See Listing 9-5 on page 9-33 for an example of using the `getRotnOp` opcode to determine if the user has selected landscape orientation.

See "Enhancing Draft-Quality Printing" on page 9-33 for more information on using the `draftBitsOp` and `noDraftBitsOp` opcodes to force the use of or to cancel the use of enhanced draft-quality printing.

Handling Printing Errors

The `PrError` function returns the result code reported by the last Printing Manager routine. The `PrSetError` procedure lets you set the value of the current Printing Manager error.

PrError

You can use the `PrError` function to get the result code returned by the last Printing Manager routine.

```
FUNCTION PrError: Integer;
```


DESCRIPTION

The `PrError` function returns the error reported by the last Printing Manager routine. If an error that does not belong to the Printing Manager occurs during the printing process, the Printing Manager puts it into low memory, where it can be retrieved with a call to `PrError`. The Printing Manager then terminates the printing loop if necessary. If you encounter an error in the middle of a printing loop, do not end printing abruptly; call the close routines for any open routines you have already made and let the Printing Manager terminate properly.

Do not display any alert or dialog boxes to report an error until the end of the printing loop. Once at the end, check for the error again; if there is no error, assume that printing completed normally. If the error is still present, then you can alert the user.

The most common error encountered is `PAPNoPrinter`, which is usually generated if no printer is selected. Since this error is so common, it is a good idea to create and display an alert box asking the user to select a printer from the Chooser when this error is encountered.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrError` function are

Trap macro	Selector
`_PrGlue`	$BA000000

RESULT CODES

iPrAbort	128	Application or user requested cancel
opNotImpl	2	Requested PrGeneral opcode not implemented in the current printer driver
noSuchRsl	1	Resolution requested with the PrGeneral procedure is not supported
noErr	0	No error
iPrSavPFil	–1	Problem saving print file
controlErr	–17	Unimplemented control instructions; the Device Manager returns this result code
iIOAbort	–27	I/O error
iMemFullErr	–108	There is not enough room in the heap zone
resNotFound	–192	The current printer driver does not support PrGeneral (described on page 9-72); you should clear this error with a call to PrSetError (described in the next section) with a parameter value of 0; otherwise, PrError might still contain this error the next time you check it

The following result codes are specific to the LaserWriter 8 printer driver:

PAPNoCCBs	–4096	There are no free connect control blocks (CCBs) available
PAPBadRefnum	–4097	Bad connection reference number
PAPActive	–4098	The request is already active
PAPTooBig	–4099	The write request is too big
PAPConnClosed	–4100	The connection is closed
PAPNoPrinter	–4101	The printer is not found, is closed, or is not selected
	–8131	Printer not responding
manualFeedTOErr	–8132	A timeout occurred (that is, no communication has occurred with the printer for two minutes); this is usually caused by an extremely long imaging time or a dropped connection
generalPSErr	–8133	A PostScript error occurred during transmission of data to the printer; this is most often caused by a bug in the application-supplied PostScript code
zoomRangeErr	–8160	The print image enlarged by the user with the Page Setup dialog box overflows the available page resolution
errBadFontKeyType	–8976	Font found in printer is not Type 1, TrueType, or bitmapped font
errPSStateUnderflow	–8977	PostScript stack underflow while restoring graphics state
errNoPattern	–8978	The pixel pattern could not be found and could not be built
errBadConverterID	–8979	The 'PDEF' converter doesn't exist

errNoPagesSpooled	–8980	Application called PrOpenDoc and PrCloseDoc without calling PrOpenPage and PrClosePage in between
errNullColorInfo	–8981	The getColor function called with null GetColorInfo handle
errPSFileNameNull	–8982	The filename pointer for the spool file is null
errSpoolFolderIsAFile	–8983	The spool folder is a file instead of a folder
errBadConverterIndex	–8984	When saving a spool file to disk, the value fileTypeIndex had no matching entry in the driver
errDidNotDownloadFont	–8985	A PostScript outline could not be found for a PostScript font, and there is no associated 'sfnt' resource
errBitmapFontMissing	–8986	Unable to build bitmap for font
errPSFileName	–8987	PostScript file isn't named
errCouldNotMakeNumberedFilename	–8989	Could not make a unique filename for the spool file
errBadSpoolFileVersion	–8990	Bad version number in header of spool file
errNoProcSetRes	–8991	The resource describing needed procedure sets is unavailable for the PostScript prolog
errInLineTimeout	–8993	The printer is not responding
errUnknownPSLevel	–8994	The PostScript level has an unknown value
errFontNotFound	–8995	Font query reply didn't match any fonts in list of PostScript names
errSizeListBad	–8996	The size list contained an entry that could not be reconciled with the typeface list
errFaceListBad	–8997	Entry could not be found in typeface list
errNotAKey	–8998	Key for desired font number and style could not be found in font table

SEE ALSO

See "Handling Printing Errors" on page 9-41 for more information on using PrError. See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about displaying alert and dialog boxes.

PrSetError

You can use the `PrSetError` procedure to set the value of the current printing error.

```
PROCEDURE PrSetError (iErr: Integer);
```

`iErr` The result to set as the current printing error.

DESCRIPTION

The `PrSetError` procedure stores the specified value into the global variable `PrintErr`, where the Printing Manager keeps its result code. You can use `PrSetError` to cancel a printing operation.

ASSEMBLY-LANGUAGE INFORMATION

You should not directly access the location of the global variable `PrintErr`; instead you should use the `PrError` function or `PrSetError` procedure to get the value of this variable.

The trap macro and routine selector for the `PrSetError` procedure are

Trap macro	Selector
`_PrGlue`	$C0000200

Low-Level Routines

Low-level routines are available for use when printing on some ImageWriter printers. (The ImageWriter LQ driver does not support these routines.) However, Apple strongly discourages you from using these routines—with the exception of the `PrDrvrVers` function. The others are documented here only for completeness.

Instead of using the low-level routines, you should use the high-level routines of the Printing Manager. Low-level routines are not guaranteed to work in precisely the same manner in future versions of the system software. Low-level routines are primarily suited for functions such as text streaming (the process of receiving data from a source and printing it immediately, without any intermediate formatting). In addition, if you use the low-level routines and the user prints a document on a LaserWriter printer, the LaserWriter printer driver translates all calls to low-level routines to the matching high-level routines, so your application does not gain a speed advantage.

▲ **WARNING**
Apple strongly discourages you from using these routines. If you do, do not mix high-level routines and low-level routines after opening the printer driver. The only exception to this is that you may use the `PrDrvrVers` function (described next) with the high-level routines. ▲

PrDrvrVers

You can use the `PrDrvrVers` function to determine the version of the printer driver for the current printer.

```
FUNCTION PrDrvrVers: Integer;
```

DESCRIPTION

The `PrDrvrVers` function returns the version number of the printer driver for the current printer. This is the only low-level printing function you may call from the high-level interface.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrDrvrVers` function are

Trap macro	Selector
`_PrGlue`	$9A000000

PrDrvrOpen

You can use the `PrDrvrOpen` procedure to open the current printer driver.

```
PROCEDURE PrDrvrOpen;
```

DESCRIPTION

The `PrDrvrOpen` procedure opens the printer driver, reading it into memory if necessary.

SPECIAL CONSIDERATIONS

Use the `PrDrvrOpen` procedure with the `PrDrvrClose` procedure (described in the next section). Do not mix these procedures with the `PrOpen` and `PrClose` procedures (described on page 9-57 and page 9-58, respectively).

Apple strongly discourages you from using this routine.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrDrvrOpen` procedure are

Trap macro	Selector
`_PrGlue`	$80000000

PrDrvrClose

You can use the `PrDrvrClose` procedure to close the printer driver.

```
PROCEDURE PrDrvrClose;
```

DESCRIPTION

The `PrDrvrClose` procedure closes the printer driver, releasing the memory it occupies.

SPECIAL CONSIDERATIONS

Use the `PrDrvrClose` procedure with the `PrDrvrOpen` procedure (described in the previous section). Do not mix these procedures with the `PrOpen` and `PrClose` procedures (described on page 9-57 and page 9-58, respectively).

Apple strongly discourages you from using this routine.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrDrvrClose` procedure are

Trap macro	Selector
`_PrGlue`	$88000000

PrDrvrDCE

You can use the `PrDrvrDCE` function to get a handle to the current printer driver's device control entry (DCE).

```
FUNCTION PrDrvrDCE: Handle;
```

DESCRIPTION

The `PrDrvrDCE` function returns a handle to the current printer driver's DCE. A printer driver's DCE contains specific information about that printer driver. You can also get a handle to the driver's DCE by calling the Device Manager function `GetDCtlEntry`.

SPECIAL CONSIDERATIONS

Apple strongly discourages you from using this routine.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrDrvrDCE` function are

Trap macro	Selector
`_PrGlue`	$94000000

SEE ALSO

For more information about DCEs and how the Device Manager uses them, see *Inside Macintosh: Devices.*

PrCtlCall

You can use the `PrCtlCall` procedure to send various requests to the current printer driver.

```
PROCEDURE PrCtlCall (iWhichCtl: Integer; lParam1: LongInt;
                     lParam2: LongInt; lParam3: LongInt);
```

`iWhichCtl` A value that indicates the operation to perform. You can use these constants in this parameter:

```
CONST
  iPrBitsCtl     = 4;  {print a bitmap object}
  iPrIOCtl       = 5;  {perform text streaming}
  iPrEvtCtl      = 6;  {print object specified in }
                       { lParam1 parameter}
  iPrDevCtl      = 7;  {device control command}
```

`lParam1` The use of this parameter varies according to the value in the `iWhichCtl` parameter. See the following paragraphs for this information.

`lParam2` The use of this parameter varies according to the value in the `iWhichCtl` parameter. See the following paragraphs for this information.

`lParam3` The use of this parameter varies according to the value in the `iWhichCtl` parameter. See the following paragraphs for this information.

DESCRIPTION

The `PrCtlCall` procedure performs the operation indicated by the `iWhichCtl` parameter. Depending on the operation, `PrCtlCall` may also use information in the `lParam1`, `lParam2`, and `lParam3` parameters. The `PrCtlCall` procedure calls the printer driver's control routine. Instead of sending the low-level calls to the printer driver, the `PrCtlCall` procedure converts the call into its high-level equivalent before execution.

You can use the `PrCtlCall` procedure with the `iPrBitsCtl` control constant when you want to print bitmaps. In this case, you should supply the parameters to `PrCtlCall` with the following information:

`iWhichCtl` The constant `iPrBitsCtl`. This constant allows you to send all or part of a QuickDraw bitmap directly to the printer.

`lParam1` A pointer to the QuickDraw bitmap to print.

`lParam2` A pointer to the rectangle you want to print.

`lParam3` The type of resolution used to print the bitmap. The LaserWriter printer driver ignores this flag. This parameter can have one of the following values:

Constant	Value	Description
`lScreenBits`	$00000000	The resolution is 80 by 72 dpi
`lPaintBits`	$00000001	The resolution is 72 by 72 dpi
`lHiScreenBits`	$00000002	The resolution is 160 by 144 dpi
`lHiPaintBits`	$00000003	The resolution is 144 by 144 dpi

You can use the `PrCtlCall` procedure with the `iPrIOCtl` control constant when you want text streaming in your application. (Text streaming is useful for fast printing of text when speed is more important than visual fidelity or formatting. It makes no use of QuickDraw.) In this case, you should supply the parameters to `PrCtlCall` with the following information:

`iWhichCtl` The constant `iPrIOCtl`. This constant causes text streaming to occur.

`lParam1` A pointer to the beginning of the text.

`lParam2` The number of bytes to transfer. The high-order word must be 0.

`lParam3` This should be 0.

You can use the `PrCtlCall` procedure with the `iPrEvtCtl` control constant for printing the screen or the frontmost window on most ImageWriter printers. (The LaserWriter printer driver does not support this call.) In this case, you should supply the parameters to `PrCtlCall` with the following information:

`iWhichCtl` The constant `iPrEvtCtl`. This constant prints the object you have selected using the `lParam1` parameter.

`lParam1` This parameter selects the object to be printed. If this value is $00000000, you want to print the screen. If this value is $00010000, you want to print the frontmost window.

lParam2 This should be NIL.

lParam3 This should be NIL.

You can use the PrCtlCall procedure with the iPrDevCtl control constant for controlling the printer device. In this case, you should supply the parameters to PrCtlCall with the following information:

iWhichCtl The constant iPrDevCtl.

lParam1 The action you want to take. The values possible for this parameter are listed in Table 9-1.

lParam2 This should be NIL.

lParam3 This should be NIL.

Table 9-1 Values for the lParam1 parameter when using the iPrDevCtl control constant

Constant	Value	Description
lPrDocOpen	$00010000	Opens the document. This is similar to the high-level routine PrOpenDoc and should be followed with a call to PrCtlCall with the iPrDetl control call and an lParam1 value of lPrDocClose.
lPrReset	$00010000	Reserved.
lPrPageClose	$00020000	Closes the page. This is similar to the high-level routine PrClosePage and should follow a call to PrCtlCall with the iPrDevCtl control call and an lParam1 value of lPrPageOpen.
lPrPageEnd	$00020000	Same as lPrPageClose.
lPrLineFeed	$00030000	Paper advance.
lPrLFStd	$0003FFFF	Carriage return with line feed. The ImageWriter printer driver causes a carriage return plus a paper feed of one-sixth of an inch. The LaserWriter printer driver moves the pen location down the page.
lPrPageOpen	$00040000	Opens the page for printing. This is similar to the high-level routine PrOpenPage and should be followed with a call to PrCtlCall with the iPrDevCtl control call and an lParam1 value of lPrPageClose.
lPrDocClose	$00050000	Closes the document. This is similar to the high-level routine PrCloseDoc and should follow a call to PrCtlCall with the iPrDevCtl control call and an lParam1 value of lPrDocOpen.

SPECIAL CONSIDERATIONS

Apple strongly discourages you from using this routine.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PrCtlCall` procedure are

Trap macro	Selector
`_PrGlue`	$A0000E00

Application-Defined Routines

The printer driver for the current printer periodically calls an idle procedure while sending a document to the printer. You can provide your own idle procedure (here called `MyDoPrintIdle`) that handles events in a dialog box reporting the status of the print job.

If you add items—such as checkboxes and radio buttons—to the default style or job dialog box, your application uses the `PrDlgMain` function to display the dialog box. In one of the parameters to `PrDlgMain`, you pass the address of an initialization function (here called `MyPrDialogAppend`) that you use to append items to your dialog box.

If you append items to the style or job dialog boxes, you need to provide a function (sometimes called a *dialog hook*) to handle events in these items. You should also provide an event filter function to handle events that the Dialog Manager doesn't handle—such as update events in background windows—in modal dialog boxes. For a description of how to handle events in dialog boxes and how to write an event filter function for modal dialog boxes, see the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

MyDoPrintIdle

The printer driver for the current printer periodically calls an idle procedure while sending a document to the printer. The Printing Manager's default idle procedure allows the user to cancel printing. This procedure polls the keyboard and sets the `iPrAbort` result code if the user presses Command-period to cancel the print job. However, the default idle procedure does not display a print status dialog box. It is up to the printer driver or your application to display a print status dialog box.

Most printer drivers display their own status dialog boxes. However, your application can display its own status dialog box that reports the current status of the printing operation to the user. If it does, your status dialog box should allow the user to press Command-period to cancel the printing operation, and it may also provide a button allowing the user to cancel the printing operation. To handle update events in your status dialog box, Command-period keyboard events, and clicks in your Cancel button (if you provide one), you should provide your own idle procedure.

```
PROCEDURE MyDoPrintIdle;
```

DESCRIPTION

As described in "Writing an Idle Procedure" beginning on page 9-38, you install your idle procedure in the `pIdleProc` field of the `TPrint` record. The printer driver runs your idle procedure periodically. It stops running once the entire document has been sent to the printer and does not run while the printer actually prints. The idle procedure takes no parameters and returns no result.

SEE ALSO

See Figure 9-9 on page 9-14 for an example of an application-defined status dialog box. Listing 9-9 on page 9-40 illustrates an idle procedure. See "Writing an Idle Procedure" beginning on page 9-38 for complete information about providing your own idle procedure.

MyPrDialogAppend

If you customize a style or job dialog box, your application uses the `PrDlgMain` function to display the dialog box. In one of the parameters to `PrDlgMain`, you pass the address of an initialization function that you use to append items—such as checkboxes and radio buttons—to the dialog box. Here is how might declare your initialization function if you were to name it `MyPrDialogAppend`:

```
FUNCTION MyPrDialogAppend (hPrint: THPrint): TPPrDlg;
```

`hPrint` A handle to a `TPrint` record (described on page 9-44).

DESCRIPTION

Your `MyPrDialogAppend` function should use the Dialog Manager procedure `AppendDITL` to add items to the style or job dialog box for the document whose `TPrint` record is passed in the `hPrint` parameter. As its function result, your function should return a pointer to the `TPrDlg` record (described on page 9-50) for the customized style or job dialog box.

You can use the `PrStlInit` or `PrJobInit` function (described on page 9-64 and page 9-65, respectively) to get an initialized `TPrDlg` record for the current printer.

Your `MyPrDialogAppend` function should install pointers to two functions in the `TPrDlg` record for this dialog box. Put a pointer to one function in the `pFltrProc` field; this function should handle events (such as update events in background applications and disk-inserted events) that the Dialog Manager doesn't handle in a modal dialog box. Put a pointer to the second function in the `pItemProc` field; this function should handle events, such as mouse clicks, in the items added to the dialog box.

SEE ALSO

Listing 9-8 on page 9-37 shows an example of the `MyPrDialogAppend` function; Listing 9-7 on page 9-37 shows how to pass the address of this function to the `PrDlgMain` function. See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about the `AppendDITL` procedure and about handling events in dialog boxes.

Summary of the Printing Manager

Pascal Summary

Constants

```
CONST
   iPrPgFst       = 1;        {page range constant--first page}
   iPrRelease     = 3;        {current version number of the printer driver}
   iPrPgFract     = 120;      {page scale factor}
   iPFMaxPgs      = 128;      {maximum pages in spool file}
   iPrPgMax       = 9999;     {page range constant--last page}

   {PrCtlCall constants for the iWhichCtl parameter}
   iPrBitsCtl     = 4;        {print a bitmap object}
   iPrIOCtl       = 5;        {perform text streaming}
   iPrEvtCtl      = 6;        {print object specified in lParam1 parameter}
   iPrDevCtl      = 7;        {device control command}

   {constants used with iPrBitsCtl (in the lParam1 parameter of PrCtlCall)}
   lScreenBits    = $00000000;   {resolution is 80 x 72 dpi}
   lPaintBits     = $00000001;   {resolution is 72 x 72 dpi}
   lHiScreenBits  = $00000002;   {resolution is 160 x 144 dpi}
   lHiPaintBits   = $00000003;   {resolution is 144 x 144 dpi}

   {constants used with iPrEvtCtl (in the lParam3 parameter of PrCtlCall)}
   lPrEvtAll      = $0002FFFD;   {the entire screen}
   lPrEvtTop      = $0001FFFD;   {the frontmost window}

   {constants used with iPrDevCtl (in the lParam1 parameter of PrCtlCall)}
   lPrReset       = $00010000;   {reserved}
   lPrLineFeed    = $00030000;   {paper advance}
   lPrLFStd       = $0003FFFF;   {carriage return with line feed}
   lPrLFSixth     = $0003FFFF;   {used for low-level call for ImageWriter}
   lPrPageEnd     = $00020000;   {end page}
   lPrDocOpen     = $00010000;   {open document for printing}
   lPrPageOpen    = $00040000;   {open page for printing}
   lPrPageClose   = $00020000;   {close page for printing}
```

```
   lPrDocClose    = $00050000;   {close document for printing}

   bDraftLoop     = 0;        {draft-quality printing}
   bSpoolLoop     = 1;        {deferred printing}
   bUser1Loop     = 2;        {reserved}
   bUser2Loop     = 3;        {reserved}

   iPrSavPFil     = -1;       {problem saving print file}
   iPrAbort       = $0080;    {the user pressed Command-period}

   iFMgrCtl    = 8;           {File Mgr's dialog-hook proc's control number}
   pPrGlobals  = $00000944;   {PrVars low memory area}

   iPrDrvrRef     = -3;       {reference number of printer driver}

   {opcodes used with PrGeneral}
   getRslDataOp   = 4;        {get resolutions for the current printer}
   setRslOp       = 5;        {set resolutions for a TPrint record}
   draftBitsOp    = 6;        {force enhanced draft-quality printing}
   noDraftBitsOp  = 7;        {cancel enhanced draft-quality printing}
   getRotnOp      = 8;        {get page orientation of a TPrint record}

   {result codes from PrGeneral}
   noSuchRsl      = 1;        {resolution not supported}
```

Data Types

```
TYPE
   TPPrint = ^TPrint;         {pointer to a TPrint record}
   THPrint = ^TPPrint;        {handle to a TPrint record}
   TPrint =                   {print record}
   RECORD
      iPrVersion: Integer;    {reserved}
      prInfo:     TPrInfo;    {resolution of device & page rectangle}
      rPaper:     Rect;       {paper rectangle}
      prStl:      TPrStl;     {printer driver number & feed type}
      prInfoPT:   TPrInfo;    {reserved}
      prXInfo:    TPrXInfo;   {reserved}
      prJob:      TPrJob;     {information from the job dialog box}
      printX:     ARRAY[1..19] OF Integer;
                              {reserved}
   END;
```

```
TPPrInfo = ^TPrInfo;
TPrInfo =                  {printer information record}
RECORD
   iDev:    Integer;       {reserved}
   iVRes:   Integer;       {vertical resolution of printer, in dpi}
   iHRes:   Integer;       {horizontal resolution of printer, in dpi}
   rPage:   Rect;          {the page rectangle}
END;

TPPrJob = ^TPrJob;
TPrJob =                      {print job record}
RECORD
   iFstPage:   Integer;       {first page of page range}
   iLstPage:   Integer;       {last page of page range}
   iCopies:    Integer;       {number of copies}
   bJDocLoop:  SignedByte;    {printing method: draft or deferred}
   fFromUsr:   Boolean;       {reserved}
   pIdleProc:  PrIdleProcPtr; {pointer to an idle procedure}
   pFileName:  StringPtr;     {spool filename: NIL for default}
   iFileVol:   Integer;       {spool file volume; set to 0 initially}
   bFileVers:  SignedByte;    {spool file version; set to 0 initially}
   bJobX:      SignedByte;    {reserved}
END;

TPPrStl = ^TPrStl;
TPrStl =                   {printing style record}
RECORD
   wDev:    Integer;       {device number of printer}
   iPageV:  Integer;       {reserved}
   iPageH:  Integer;       {reserved}
   bPort:   SignedByte;    {reserved}
   feed:    TFeed;         {feed type}
END;

TPPrStatus = ^TPrStatus;
TPrStatus =                {printing status record}
RECORD
   iTotPages:  Integer;    {total pages in print file}
   iCurPage:   Integer;    {current page number}
   iTotCopies: Integer;    {total copies requested}
   iCurCopy:   Integer;    {current copy number}
   iTotBands:  Integer;    {reserved}
   iCurBand:   Integer;    {reserved}
   fPgDirty:   Boolean;    {TRUE if current page has been written to}
```

```
   fImaging:   Boolean;    {reserved}
   hPrint:     THPrint;    {handle to the active TPrint record}
   pPrPort:    TPPrPort;   {pointer to the active printing graphics port}
   hPic:       PicHandle;  {handle to the active picture}
END;

TPPrDlg = ^TPrDlg;
TPrDlg =                            {print dialog box record}
RECORD
   Dlg:        DialogRecord;        {a dialog record}
   pFltrProc:  ModalFilterProcPtr;  {pointer to event filter}
   pItemProc:  PItemProcPtr;        {pointer to item-handling function}
   hPrintUsr:  THPrint;             {handle to a TPrint record}
   fDoIt:      Boolean;             {TRUE means user clicked OK}
   fDone:      Boolean;             {TRUE means user clicked OK or Cancel}
   lUser1:     LongInt;             {storage for your application}
   lUser2:     LongInt;             {storage for your application}
   lUser3:     LongInt;             {storage for your application}
   lUser4:     LongInt;             {storage for your application}
END;

TPPrPort = ^TPrPort;
TPrPort =                 {printing graphics port record}
RECORD
   gPort:      GrafPort;   {graphics port for printing}
   gProcs:     QDProcs;    {procedures for printing in the graphics port}
   lGParam1:   LongInt;    {reserved}
   lGParam2:   LongInt;    {reserved}
   lGParam3:   LongInt;    {reserved}
   lGParam4:   LongInt;    {reserved}
   fOurPtr:    Boolean;    {reserved}
   fOurBits:   Boolean;    {reserved}
END;

TFeed = (feedCut,feedFanfold,feedMechCut,feedOther);

TScan = (scanTB,scanBT,scanLR,scanRL);

TPRect = ^Rect;

PrIdleProcPtr = ProcPtr;

PItemProcPtr = ProcPtr;
```

```
   PDlgInitProcPtr = ProcPtr;

{records used by PrGeneral}

   TGnlData =
   RECORD
      iOpCode:    Integer; {opcode passed to PrGeneral}
      iError:     Integer; {result code returned by PrGeneral}
      lReserved:  LongInt; {more fields here depending on opcode}
   END;

   TGetRslBlk =               {get-resolution record}
   RECORD
      iOpCode:    Integer;    {the getRslDataOp opcode}
      iError:     Integer;    {result code returned by PrGeneral}
      lReserved:  LongInt;    {reserved}
      iRgType:    Integer;    {printer driver version number}
      xRslRg:     TRslRg;     {x-direction resolution range}
      yRslRg:     TRslRg;     {y-direction resolution range}
      iRslRecCnt: Integer;    {number of resolution records}
      rgRslRec:   ARRAY[1..27] OF TRslRec;
                              {array of resolution records}
   END;

   TRslRg =
   RECORD
      iMin:    Integer; {minimum resolution supported}
      iMax:    Integer; {maximum resolution supported}
   END;

   TRslRec =
   RECORD
      iXRsl:   Integer; {discrete resolution, x direction}
      iYRsl:   Integer; {discrete resolution, y direction}
   END;

   TSetRslBlk =               {set-resolution record}
   RECORD
      iOpCode:    Integer;    {the setRslOp opcode}
      iError:     Integer;    {result code returned by PrGeneral}
      lReserved:  LongInt;    {reserved}
      hPrint:     THPrint;    {handle to the current TPrint record}
      iXRsl:      Integer;    {x-direction resolution you want}
      iYRsl:      Integer;    {y-direction resolution you want}
   END;
```

```
TDftBitsBlk =              {draft bits record}
RECORD
   iOpCode:    Integer;    {draftBitsOp or noDraftBitsOp opcode}
   iError:     Integer;    {result code returned by PrGeneral}
   lReserved:  LongInt;    {reserved}
   hPrint:     THPrint;    {handle to the current TPrint record}
END;

TGetRotnBlk =              {page orientation record}
RECORD
   iOpCode:    Integer;    {the getRotnOp opcode}
   iError:     Integer;    {result code returned by PrGeneral}
   lReserved:  LongInt;    {reserved}
   hPrint:     THPrint;    {handle to current TPrint record}
   fLandscape: Boolean;    {TRUE if user selected landscape printing}
   bXtra:      SignedByte; {reserved}
END;
```

Printing Manager Routines

Opening and Closing the Printing Manager

```
PROCEDURE PrOpen;
PROCEDURE PrClose;
```

Initializing and Validating TPrint Records

```
PROCEDURE PrintDefault      (hPrint: THPrint);
FUNCTION PrValidate         (hPrint: THPrint): Boolean;
```

Displaying and Customizing the Print Dialog Boxes

```
FUNCTION PrStlDialog        (hPrint: THPrint): Boolean;
FUNCTION PrJobDialog        (hPrint: THPrint): Boolean;
FUNCTION PrDlgMain          (hPrint: THPrint; pDlgInit: PDlgInitProcPtr):
                             Boolean;
FUNCTION PrStlInit          (hPrint: THPrint): TPPrDlg;
FUNCTION PrJobInit          (hPrint: THPrint): TPPrDlg;
PROCEDURE PrJobMerge        (hPrintSrc: THPrint; hPrintDst: THPrint);
```

Printing a Document

```
FUNCTION PrOpenDoc          (hPrint: THPrint; pPrPort: TPPrPort;
                             pIOBuf: Ptr): TPPrPort;
PROCEDURE PrCloseDoc        (pPrPort: TPPrPort);
PROCEDURE PrOpenPage        (pPrPort: TPPrPort; pPageFrame: TPRect);
PROCEDURE PrClosePage       (pPrPort: TPPrPort);
PROCEDURE PrPicFile         (hPrint: THPrint; pPrPort: TPPrPort;
                             pIOBuf: Ptr; pDevBuf: Ptr;
                             VAR prStatus: TPrStatus);
```

Optimizing Printing

```
PROCEDURE PrGeneral         (pData: Ptr);
```

Handling Printing Errors

```
FUNCTION PrError            : Integer;
PROCEDURE PrSetError        (iErr: Integer);
```

Low-Level Routines

```
FUNCTION PrDrvrVers         : Integer;
PROCEDURE PrDrvrOpen;
PROCEDURE PrDrvrClose;
FUNCTION PrDrvrDCE          : Handle;
PROCEDURE PrCtlCall         (iWhichCtl: Integer; lParam1: LongInt;
                             lParam2: LongInt; lParam3: LongInt);
```

Application-Defined Routines

```
PROCEDURE MyDoPrintIdle;
FUNCTION MyPrDialogAppend   (hPrint: THPrint): TPPrDlg;
```

C Summary

Constants

```
enum {
   iPrPgFst    = 1,     /* page range constant--first page */
   iPrRelease  = 3,     /* current version number of the printer driver */
   iPrPgFract  = 120,   /* page scale factor */
   iPFMaxPgs   = 128,   /* maximum pages in spool file */
   iPrPgMax    = 9999,  /* page range constant--last page */

   /* PrCtlCall constants for the iWhichCtl parameter */
   iPrBitsCtl     = 4,  /* print a bitmap object */
   iPrIOCtl       = 5,  /* perform text streaming */
   iPrEvtCtl      = 6,  /* print object specified in lParam1 parameter */
   iPrDevCtl      = 7,  /* device control command */

   /* constants used with iPrBitsCtl (in the lParam1 parameter
      of PrCtlCall) */
   lScreenBits    = 0,           /* resolution is 80 x 72 dpi */
   lPaintBits     = 1,           /* resolution is 72 x 72 dpi */
   lHiScreenBits  = 0x00000002,  /* resolution is 160 x 144 dpi */
   lHiPaintBits   = 0x00000003,  /* resolution is 144 x 144 dpi */

   /* constants used with iPrEvtCtl (in the lParam3 parameter
      of PrCtlCall) */
   lPrEvtAll      = 0x0002FFFD,  /* the entire screen */
   lPrEvtTop      = 0x0001FFFD,  /* the frontmost window */

   /* constants used with iPrDevCtl (in the lParam1 parameter
      of PrCtlCall) */
   lPrReset       = 0x00010000,  /* reserved */
   lPrLineFeed    = 0x00030000,  /* paper advance */
   lPrLFStd       = 0x0003FFFF,  /* carriage return with line feed */
   lPrLFSixth     = 0x0003FFFF,  /* used for low-level call for
                                    ImageWriter */
   lPrPageEnd     = 0x00020000,  /* end page */
   lPrDocOpen     = 0x00010000,  /* open document for printing */
   lPrPageOpen    = 0x00040000,  /* open page for printing */
   lPrPageClose   = 0x00020000,  /* close page for printing */
   lPrDocClose    = 0x00050000,  /* close document for printing */
```

```
   bDraftLoop     = 0,           /* draft-quality printing */
   bSpoolLoop     = 1,           /* deferred printing */
   bUser1Loop     = 2,           /* reserved */
   bUser2Loop     = 3,           /* reserved */

   iPrSavPFil     = -1,          /* problem saving print file */
   iPrAbort       = 0x0080,      /* the user pressed Command-period */

   iFMgrCtl       = 8,           /* File Mgr's dialog-hook proc's control
                                    number */
   pPrGlobals     = 0x00000944,     /* PrVars low memory area */

   iPrDrvrRef     = -3, /* reference number of printer driver */

   /* opcodes used with PrGeneral */
   getRslDataOp   = 4,  /* get resolutions for the current printer */
   setRslOp       = 5,  /* set resolutions for a TPrint record */
   draftBitsOp    = 6,  /* force enhanced draft-quality printing */
   noDraftBitsOp  = 7,  /* cancel enhanced draft-quality printing */
   getRotnOp      = 8,  /* get page orientation of a TPrint record */

   /* result code from PrGeneral */
   noSuchRsl      = 1,  /* resolution not supported */
};
```

Data Types

```
struct TPrint {         /* print record */
   short    iPrVersion; /* reserved */
   TPrInfo  prInfo;     /* resolution of device & page rectangle */
   Rect     rPaper;     /* paper rectangle */
   TPrStl   prStl;      /* printer driver number & feed type */
   TPrInfo  prInfoPT;   /* reserved */
   TPrXInfo prXInfo;    /* reserved */
   TPrJob   prJob;      /* information from the job dialog box */
   short    printX[19]; /* reserved */
};
typedef struct TPrint TPrint;
typedef TPrint *TPPrint, **THPrint;

struct TPrInfo {        /* printer information record */
   short    iDev;       /* reserved */
   short    iVRes;      /* vertical resolution of printer, in dpi */
```

```
   short    iHRes;      /* horizontal resolution of printer, in dpi */
   Rect     rPage;      /* the page rectangle */
};
typedef struct TPrInfo TPrInfo;
typedef TPrInfo *TPPrInfo;

struct TPrJob {         /* print job record */
   short    iFstPage;   /* first page of page range */
   short    iLstPage;   /* last page of page range */
   short    iCopies;    /* number of copies */
   char     bJDocLoop;  /* printing method: draft or deferred */
   Boolean  fFromUsr;   /* reserved */
   PrIdleProcPtr
            pIdleProc;  /* pointer to an idle procedure */
   StringPtr
            pFileName;  /* spool filename: NIL for default */
   short    iFileVol;   /* spool file volume; set to 0 initially */
   char     bFileVers;  /* spool file version; set to 0 initially */
   char     bJobX;      /* reserved */
};
typedef struct TPrJob TPrJob;
typedef TPrJob *TPPrJob;

struct TPrStl {      /* printing style record */
   short    wDev;    /* device number of printer */
   short    iPageV;  /* reserved */
   short    iPageH;  /* reserved */
   char     bPort;   /* reserved */
   TFeed    feed;    /* feed type */
};
typedef struct TPrStl TPrStl;
typedef TPrStl *TPPrStl;

struct TPrStatus {         /* printing status record */
   short       iTotPages;  /* total pages in print file */
   short       iCurPage;   /* current page number */
   short       iTotCopies; /* total copies requested */
   short       iCurCopy;   /* current copy number */
   short       iTotBands;  /* reserved */
   short       iCurBand;   /* reserved */
   Boolean     fPgDirty;   /* TRUE if current page has been written to */
   Boolean     fImaging;   /* reserved */
   THPrint     hPrint;     /* handle to the active TPrint record */
```

```
   TPPrPort    pPrPort;    /* pointer to the active printing graphics port */
   PicHandle   hPic;       /* handle to the active picture */
};
typedef struct TPrStatus TPrStatus;
typedef TPrStatus *TPPrStatus;

struct TPrDlg {              /* print dialog box record */
   DialogRecord   Dlg;       /* a dialog record */
   ModalFilterProcPtr
                  pFltrProc; /* pointer to event filter */
   PItemProcPtr   pItemProc; /* pointer to item-handling function */
   THPrint        hPrintUsr; /* handle to a TPrint record */
   Boolean        fDoIt;     /* TRUE means user clicked OK */
   Boolean        fDone;     /* TRUE means user clicked OK or Cancel */
   long           lUser1;    /* storage for your application */
   long           lUser2;    /* storage for your application */
   long           lUser3;    /* storage for your application */
   long           lUser4;    /* storage for your application */
};
typedef struct TPrDlg TPrDlg;
typedef TPrDlg *TPPrDlg;

typedef pascal TPPrDlg (*PDlgInitProcPtr)(THPrint hPrint);

truct TPrPort {         /* printing graphics port record */
   GrafPort gPort;      /* graphics port for printing */
   QDProcs  gProcs;     /* procedures for printing in the graphics port */
   long     lGParam1;   /* reserved */
   long     lGParam2;   /* reserved */
   long     lGParam3;   /* reserved */
   long     lGParam4;   /* reserved */
   Boolean  fOurPtr;    /* reserved */
   Boolean  fOurBits;   /* reserved */
};
typedef struct TPrPort TPrPort;
typedef TPrPort *TPPrPort;

enum {feedCut,feedFanfold,feedMechCut,feedOther};
typedef unsigned char TFeed;

enum {scanTB,scanBT,scanLR,scanRL};
typedef unsigned char TScan;

typedef Rect *TPRect;
```

```
typedef pascal void (*PrIdleProcPtr)(void);

typedef pascal void (*PItemProcPtr)(DialogPtr theDialog, short item);

/* structures used by PrGeneral */

struct TGnlData {
   short iOpCode;    /* opcode passed to PrGeneral */
   short iError;     /* result code returned by PrGeneral */
   long  lReserved;  /* more fields here depending on call */
};
typedef struct TGnlData TGnlData;

struct TGetRslBlk {        /* get-resolution record */
   short    iOpCode;       /* the getRslDataOp opcode */
   short    iError;        /* result code returned by PrGeneral */
   long     lReserved;     /* reserved */
   short    iRgType;       /* printer driver version number */
   TRslRg   xRslRg;        /* x-direction resolution range */
   TRslRg   yRslRg;        /* y-direction resolution range */
   short    iRslRecCnt;    /* number of resolution records */
   TRslRec  rgRslRec[27];  /* array of resolution records */
};
typedef struct TGetRslBlk TGetRslBlk;

struct TRslRg {
   short    iMin;          /* minimum resolution supported */
   short    iMax;          /* maximum resolution supported */
};
typedef struct TRslRg TRslRg;

struct TRslRec {
   short    iXRsl;         /* discrete resolution, x direction */
   short    iYRsl;         /* discrete resolution, y direction */
};
typedef struct TRslRec TRslRec;
```

```
struct TSetRslBlk {        /* set-resolution record */
   short    iOpCode;       /* the setRslOp opcode */
   short    iError;        /* result code returned by PrGeneral */
   long     lReserved;     /* reserved */
   THPrint  hPrint;        /* handle to the current TPrint record */
   short    iXRsl;         /* x-direction resolution you want */
   short    iYRsl;         /* y-direction resolution you want */
};
typedef struct TSetRslBlk TSetRslBlk;

struct TDftBitsBlk {       /* draft bits record */
   short    iOpCode;       /* draftBitsOp or noDraftBitsOp opcode */
   short    iError;        /* result code returned by PrGeneral */
   long     lReserved;     /* reserved */
   THPrint  hPrint;        /* handle to the current TPrint record */
};
typedef struct TDftBitsBlk TDftBitsBlk;

struct TGetRotnBlk {       /* page orientation record */
   short    iOpCode;       /* the getRotnOp opcode */
   short    iError;        /* result code returned by PrGeneral */
   long     lReserved;     /* reserved */
   THPrint  hPrint;        /* handle to current TPrint record */
   Boolean  fLandscape;    /* TRUE if user selected landscape printing */
   char     bXtra;         /* reserved */
};
typedef struct TGetRotnBlk TGetRotnBlk;
```


Printing Manager Functions

Opening and Closing the Printing Manager

```
pascal void PrOpen          (void);
pascal void PrClose         (void);
```

Initializing and Validating TPrint Records

```
pascal void PrintDefault    (THPrint hPrint);
pascal Boolean PrValidate   (THPrint hPrint);
```

Displaying and Customizing the Print Dialog Boxes

```
pascal Boolean PrStlDialog  (THPrint hPrint);
pascal Boolean PrJobDialog  (THPrint hPrint);
pascal Boolean PrDlgMain    (THPrint hPrint, PDlgInitProcPtr pDlgInit);
pascal TPPrDlg PrStlInit    (THPrint hPrint);
pascal TPPrDlg PrJobInit    (THPrint hPrint);
pascal void PrJobMerge      (THPrint hPrintSrc, THPrint hPrintDst);
```

Printing a Document

```
pascal TPPrPort PrOpenDoc   (THPrint hPrint, TPPrPort pPrPort, Ptr pIOBuf);
pascal void PrCloseDoc      (TPPrPort pPrPort);
pascal void PrOpenPage      (TPPrPort pPrPort, TPRect pPageFrame);
pascal void PrClosePage     (TPPrPort pPrPort);
pascal void PrPicFile       (THPrint hPrint, TPPrPort pPrPort, Ptr pIOBuf,
                             Ptr pDevBuf, TPrStatus *prStatus);
```

Optimizing Printing

```
pascal void PrGeneral       (Ptr pData);
```

Handling Printing Errors

```
pascal short PrError        (void);
pascal void PrSetError      (short iErr);
```

Low-Level Functions

```
pascal short PrDrvrVers     (void);
pascal void PrDrvrOpen      (void);
pascal void PrDrvrClose     (void);
pascal Handle PrDrvrDCE     (void);
pascal void PrCtlCall       (short iWhichCtl, long lParam1,
                             long lParam2, long lParam3);
```

Application-Defined Functions

```
pascal void MyDoPrintIdle   (void);
pascal TPPrDlg MyPrDialogAppend
                            (THPrint hPrint);
```

Assembly-Language Summary

Data Structures

TPrint Data Structure

0	iPrVersion	word	printer driver version that initialized this record
2	prInfo	14 bytes	TPrInfo data structure with resolution of device and page rectangle
16	rPaper	8 bytes	paper rectangle
24	prStl	8 bytes	TPrStl data structure with printer driver number and feed type
32	prInfoPT	14 bytes	TPrInfo data structure; reserved
46	prXInfo	16 bytes	TPrXInfo data structure; reserved
62	prJob	20 bytes	TPrJob data structure with printing information from the job dialog box
82	printX	40 bytes	reserved

TPrInfo Data Structure

0	iDev	6 bytes	first word is reserved; second word contains vertical resolution of printer, in dpi; third word contains horizontal resolution of printer, in dpi
6	rPage	8 bytes	the page rectangle

TPrJob Data Structure

0	iFstPage	word	first page of page range
2	iLstPage	word	last page of page range
4	iCopies	word	number of copies
6	bJDocLoop	1 byte	printing method: draft or deferred
7	fFromApp	1 byte	reserved
8	pIdleProc	long	pointer to an idle procedure
12	pFileName	long	spool filename: NIL for default
16	iFileVol	word	spool file volume: set to 0 initially
18	bFileVers	1 byte	spool file version: set to 0 initially

TPrStl Data Structure

0	wDev	word	device number of the current printer (in the high-order byte of this field); the low-order byte of this field is reserved
2	iPageV	word	reserved
4	iPageH	word	reserved
7	feed	1 byte	paper feed type

TPrStatus Data Structure

0	iTotPages	word	total pages in print file
2	iCurPage	word	current page number
4	iTotCopies	word	total copies requested
6	iCurCopy	word	current copy number
8	iTotBands	word	reserved
10	iCurBand	word	reserved
12	fPgDirty	1 byte	TRUE if current page has been written to
13	fImaging	1 byte	reserved
14	hPrint	long	handle to the active TPrint data structure
18	pPrPort	long	pointer to the active printing graphics port
22	hPic	long	handle to the active picture

TPrDlg Data Structure

0	dlg	168 bytes	a dialog record
168	pFltrProc	long	pointer to event filter
172	pItemProc	long	pointer to item-handling function
176	hPrintUsr	long	handle to TPrint data structure
180	fDoIt	1 byte	TRUE means user clicked OK
181	fDone	1 byte	TRUE means user clicked OK or Cancel
182	lUser1	long	storage for your application
186	lUser2	long	storage for your application
190	lUser3	long	storage for your application
194	lUser4	long	storage for your application

TPrPort Data Structure

0	gPort	178 bytes	graphics port and procedures for printing

Trap Macros

Trap Macros Requiring Routine Selectors

_PrGlue

Selector	Routine
$C8000000	PrOpen
$D0000000	PrClose
$20040480	PrintDefault
$52040498	PrValidate
$2A040484	PrStlDialog
$32040488	PrJobDialog
$4A040894	PrDlgMain
$3C04040C	PrStlInit
$44040410	PrJobInit
$5804089C	PrJobMerge
$04000C00	PrOpenDoc
$08000484	PrCloseDoc
$10000808	PrOpenPage
$1800040C	PrClosePage
$60051480	PrPicFile
$BA000000	PrError
$C0000200	PrSetError
$70070480	PrGeneral
$94000000	PrDrvrDCE
$9A000000	PrDrvrVers
$80000000	PrDrvrOpen
$88000000	PrDrvrClose
$A0000E00	PrCtlCall

Global Variable

PrintErr	Printing error.

Result Codes

iPrAbort	128	Application or user requested cancel
opNotImpl	2	Requested PrGeneral opcode not implemented in the current printer driver
noSuchRsl	1	Resolution requested with the PrGeneral procedure is not supported
noErr	0	No error
iPrSavPFil	–1	Problem saving print file
controlErr	–17	Unimplemented control instructions; the Device Manager returns this result code
iIOAbort	–27	I/O error
iMemFullErr	–108	There is not enough room in the heap zone
resNotFound	–192	The current printer driver does not support PrGeneral; you should clear this error with a call to PrSetError with a parameter value of 0; otherwise, PrError might still contain this error next time you check it
PAPNoCCBs	–4096	There are no free connect control blocks (CCBs) available
PAPBadRefnum	–4097	Bad connection reference number
PAPActive	–4098	The request is already active
PAPTooBig	–4099	The write request is too big
PAPConnClosed	–4100	The connection is closed
PAPNoPrinter	–4101	The printer is not found, is closed, or is not selected
	–8131	Printer not responding
manualFeedTOErr	–8132	A timeout occurred (that is, no communication has occurred with the printer for two minutes); this is usually caused by an extremely long imaging time or a dropped connection
generalPSErr	–8133	A PostScript error occurred during transmission of data to the printer; this is most often caused by a bug in the application-supplied PostScript code
zoomRangeErr	–8160	The print image enlarged by the user with the Page Setup dialog box overflows the available page resolution
errBadFontKeyType	–8976	Font found in printer is not Type 1, TrueType, or bitmapped font
errPSStateUnderflow	–8977	PostScript stack underflow while restoring graphics state
errNoPattern	–8978	The pixel pattern could not be found and could not be built
errBadConverterID	–8979	The 'PDEF' converter doesn't exist
errNoPagesSpooled	–8980	Application called PrOpenDoc and PrCloseDoc without calling PrOpenPage and PrClosePage in between
errNullColorInfo	–8981	The getColor function called with null GetColorInfo handle
errPSFileNameNull	–8982	The filename pointer for the spool file is null
errSpoolFolderIsAFile	–8983	The spool folder is a file instead of a folder

errBadConverterIndex	–8984	When saving a spool file to disk, the value `fileTypeIndex` field had no matching entry in the driver
errDidNotDownloadFont	–8985	A PostScript outline could not be found for a PostScript font, and there is no associated `'sfnt'` resource
errBitmapFontMissing	–8986	Unable to build bitmap for font
errPSFileName	–8987	PostScript file isn't named
errCouldNotMakeNumberedFilename	–8989	Could not make a unique filename for the spool file
errBadSpoolFileVersion	–8990	Bad version number in header of spool file
errNoProcSetRes	–8991	The resource describing needed procedure sets is unavailable for the PostScript prolog
errInLineTimeout	–8993	The printer is not responding
errUnknownPSLevel	–8994	The PostScript level has an unknown value
errFontNotFound	–8995	Font query reply didn't match any fonts in list of PostScript names
errSizeListBad	–8996	The size list contained an entry that could not be reconciled with the typeface list
errFaceListBad	–8997	Entry could not be found in typeface list
errNotAKey	–8998	Key for desired font number and style could not be found in font table