

# 13

Version 1.1

## PRINTING

### Includes Demonstration Program PrintingPascal

#### The Printing Manager

---

The Printing Manager is a collection of system software routines that your application can use to print to any type of connected printer using the same QuickDraw routines that your application uses for screen display. When printing, your application calls the same Printing Manager routines regardless of the type of printer selected by the user.

You can use the Printing Manager to:

- Print documents.
- Display and alter printing dialog boxes.
- Handle printing errors.

To use the Printing Manager, you must first initialise QuickDraw, the Font Manager, the Window Manager, the Menu Manager, TextEdit, and the Dialog Manager.

#### Printer Drivers

---

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. The **current printer**<sup>1</sup> is the printer driver that actually implements the routines defined by the Printing Manager.

#### Types and Characteristics of Printer Drivers

---

In general, there are two types of types of printer driver:

- QuickDraw printer drivers.
- PostScript printer drivers.

---

<sup>1</sup>The current printer is the printer that the user last selected from the Chooser.

## QuickDraw Printer Drivers

---

QuickDraw printer drivers render images using QuickDraw and then send the rendered images to the printer as bitmaps or pixel maps. Since they rely on the rendering capabilities of the Macintosh computer, QuickDraw printers are not required to have any intelligent rendering capabilities. Instead, they simply accept instructions from the printer driver to place dots on the page in specified places.

A QuickDraw printer captures the image of an entire page either in memory or in a temporary disk file known as a **spool file**, translates the pixels into dot placement instructions, and sends these instructions to the printer.

Given that over 7 million pixels are required to render an 8-by-10-inch image at 300 dots per inch, QuickDraw printers are relatively slow; accordingly, many QuickDraw printers use some form of data compression to improve their performance. The large memory requirements involved in printing to a colour printer using 8 bits per pixel may require the driver to process the image in horizontal strips, which further impairs printing speed.

## PostScript Printers

---

Unlike QuickDraw printers, PostScript printers have their own rendering capabilities. Instead of rendering the entire page on the Macintosh computer and sending all the pixels to the printer, PostScript printer drivers convert QuickDraw operations into equivalent PostScript operations and send the resulting drawing commands directly to the printer. The printer then renders the images by interpreting these commands. In this way, image processing is offloaded from the computer.

Whereas QuickDraw printer drivers must capture an entire page before sending any of it to the printer, PostScript printer drivers are able to send commands as soon as they are generated. Although this results in faster printing, it does not allow the driver to examine entire pages for their use of colour, fonts, or other resources that the printer needs to have specially processed. Accordingly, some PostScript printer drivers may capture page images in a spool file so that the driver can analyse the pages before sending them to the printer.

## Background Printing, Deferred Printing, and Spool Files

---

Some printer drivers allow users to specify **background printing**, which allows a user to work with an application while documents are printing in the background. These printer drivers send printing data to a spool file in the PrintMonitor Documents folder in the System Folder.

Some QuickDraw printer drivers provide two methods of printing documents: **deferred printing** and draft-quality. 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. 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 printer driver writes to a spool file. `PrPicFile` is then used to instruct these drivers to turn the QuickDraw picture into bit images and send them to the printer.

Do not confuse the different uses of spool files. With background printing, print files are spooled to disk so that the user can work with an application while documents are printing. You do not need to use `PrPicFile` to send these spool files to the printer — in fact, there is no reliable way to determine whether a printer driver is using a spool file for background printing. A spool file created by a printer driver using deferred printing is another matter. (As will be seen, you can readily determine whether a printer driver is using deferred printing.)

## Printer Drivers and Picture Comments

---

For most applications, sending QuickDraw's picture-drawing routines to the printer driver is sufficient. However, some applications may rely on printer drivers to provide several features (for example, rotated text or dashed lines) which are not available, or which are difficult to achieve, using QuickDraw. If your application requires these features, 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 approximations of those features.

Picture comments are data or commands, created with the QuickDraw routine `PicComment`, used for special processing by output devices such as printer drivers. They may be included in the code an application sends to a printer driver or they may be stored in the definition of a picture.

## Printer Resolution

---

Resolution is usually specified in dots-per-inch (dpi) in the x and y directions.

A printer driver supports either **discrete resolution** or **variable resolution**. If a printer driver supports discrete resolution, an application can choose from only a limited number of resolutions pre-defined by the printer driver. 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.

## Page and Paper Rectangles

---

When printing a document, you should consider the physical size of the paper and the area of the paper that the printer can use to format the document. This is usually smaller than the physical sheet of paper, generally because of the mechanical limitations of the printer.

### Page Rectangle

---

The **page rectangle** (see Fig 1) represents the boundaries of the printable area of the page. Its upper-left coordinates are always (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 (see below). For example, the lower-right corner of a page rectangle used by the PostScript LaserWriter printer driver for an 8.5-by-11-inch page is (730,552) at 72 dpi.

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.

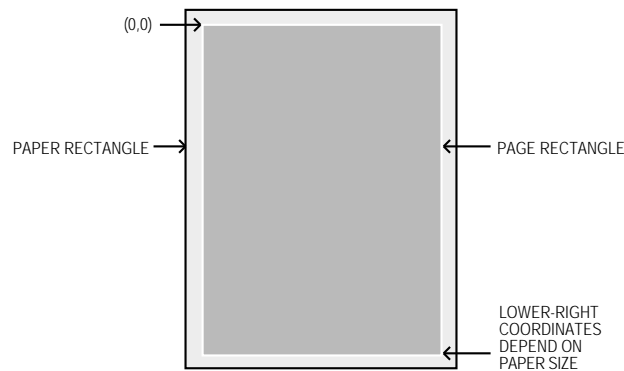


FIG 1 - PAPER AND PAGE RECTANGLES

### Paper Rectangle

---

The **paper rectangle** (see Fig 1) 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.

## Job Dialog Box, Style Dialog Box, and the TPrint Record

### Job Dialog Box and Style Dialog Box

If it is likely that the user will want to print the data created with your application, you should support both the Page Setup... command and the Print... command in your application's File menu.

In response to the Page Setup... command, your application should display the current printer's **style dialog box**, which allows the user to specify printing options, such as paper size and printing orientation, that your application needs for formatting the document in the frontmost window. Each printer driver defines its own style dialog box. Fig 2 shows the style dialog box for the Color StyleWriter 2500 printer.

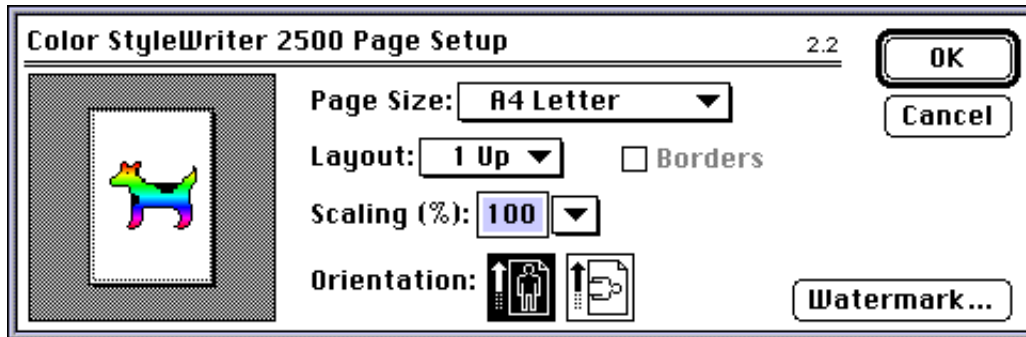


FIG 2 - STYLE DIALOG BOX FOR STYLEWRITER II PRINTER

In response to the Print... command, your application should display the current printer's **job dialog box**, which solicits printing information from the user (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 job dialog box. Fig 3 shows the job dialog box for the Color StyleWriter 2500 printer.

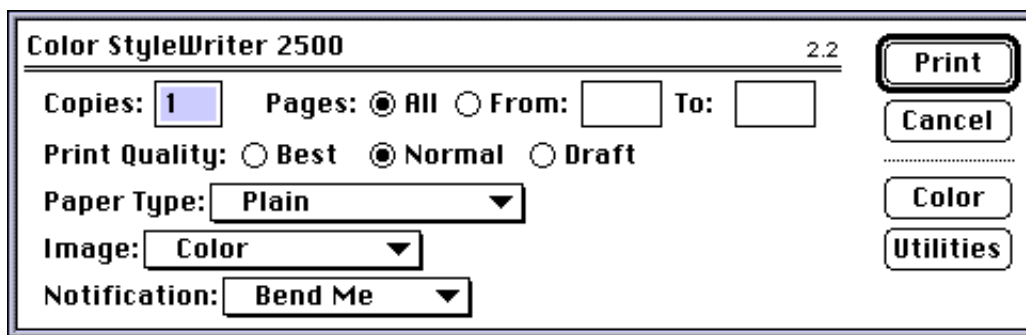


FIG 3 - JOB DIALOG BOX FOR STYLEWRITER II PRINTER

Note that many applications add items to the basic style and job dialog boxes so as to provide the user with additional control over printing operations within that application.

### Preserving the User's Printing Preferences

The only information you should preserve each time the user prints the document should be that obtained via the style dialog box. The information supplied by the user through the job dialog box should pertain to the document only while the document prints, and you should not re-use this information if the user prints the document again.

A TPrint record (see below) stores information about the user's choices made via the style (and the job) dialog box. Thus you can preserve the information obtained via the style dialog box by saving the TPrint record associated with a document in that document's data or resource fork.

The values specified by the user through the style dialog box apply only to the printing of the document in the active window. In general, the user should have to specify these values only once per document (although the user can, of course, choose to change the settings at any time).

## **Displaying the Style and Job Dialog Boxes**

`PrStlDialog` is used to display the style dialog box defined by the resource file for the current printer. `PrJobDialog` is used to display the job dialog box defined by the resource file for the current printer. These functions handle all user interaction in the items defined by the printer driver until the user clicks the OK or Cancel button. You must call `PrOpen` before calling `PrStlDialog` because the current printer driver must be open for your application to successfully call `PrStlDialog`.

## **Customising the Style and Job Dialog Boxes**

If you wish to customise the style and/or job dialog boxes so as to solicit additional information from the user, you must provide a function that handles events such as mouse clicks in any items that you add to the dialog box. You must also provide an event filter function to handle events not handled by the Dialog Manager in a modal dialog box.

Note that `PrDlgMain`, not `PrStlDialog` and `PrJobDialog`, is used to display a customised style or job dialog box

## **The TPrint Record**

To print a document, you need to create a **print record**. The `TPrint` record is a data structure of type `TPrint`. 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 itself, using `NewHandle`, and then initialises the `TPrint` record using `PrintDefault`. Your application may also use an existing `TPrint` record, in which case you can validate the record using `PrValidate`. (`PrValidate` checks all fields of the `TPrint` record to ensure compatibility with the current printer.)

When the user chooses the **Print...** command, your application passes a handle to a `TPrint` record to `PrJobDialog` (or `PrDlgMain` in the case of customised job dialog boxes) to display a job dialog box to the user. `PrJobDialog` (or `PrDlgMain`) alters the `prJob` field (a `TPrJob` record) of 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 `PrStlDialog` (or `PrDlgMain` in the case of customised style dialog boxes) to display a style dialog box to the user. `PrStlDialog` (or `PrDlgMain`) alters the `prInfo` field (a `TPrInfo` record) of the `TPrint` record according to the user's responses.

The `TPrint` record, including its constituent `TPrJob` and `TPrInfo` records, is shown at Fig 4. Note also the `prInfo` field (a `TPrInfo` record), which contains resolution and page rectangle information.

```

TPrint = record
iPrVersion: integer; { (Reserved)}
prInfo: TPrInfo;      { PrInfo data associated with the current style.}
rPaper: Rect;          { Paper rectangle (offset from rPage).}
prStl: TPrStl;         { This print request's style.}
prInfoPT: TPrInfo;     { (Reserved)}
prXInfo: TPrXInfo;     { (Reserved)}
prJob: TPrJob;          { Print Job request.}
case integer of
0: (
printX: array [1..19] of integer; { (Reserved)}
);
1: (
prFlag1: TPrFlag1;
iZoomMin: integer;
iZoomMax: integer;
hDocName: StringHandle;
);
end;

TPPrint = ^TPrint;
THPrint = ^TPPrint;

```

NOTE: Some printer drivers always set the **iCopies** field to 1, regardless of the user's entry in the job dialog box, and handle multiple copies internally.

```

TPrJob = record
iFstPage: integer; { First page of page range.}
iLstPage: integer; { Last page of page range.}
Copies: integer;   { Number of copies.}
bJDocLoop: SInt8;  { Printing method - draft or deferred.}
fFromUsr: boolean; { (Reserved)}
pIdleProc: PrIdleUPP; { Pointer to an idle procedure.}
pFileName: StringPtr; { Spool file name: NIL for default.}
iFileVol: integer;  { Spool file volume, set to 0 initially}
bFileVers: SInt8;   { Spool file version, set to 0 initially}
bJobX: SInt8;       { (Reserved)}
end;

```

TPPrJob = ^TPrJob;

```

TPrStl = record
wDev: integer; { Device number of printer.}
iPageV: integer; { (Reserved)}
iPageH: integer; { (Reserved)}
bPort: SInt8;   { (Reserved)}
feed: TFeed;    { Feed type.}
end;

```

TPPrStl = ^TPrStl;

```

TPrInfo = record
iDev: integer; { (Reserved)}
iVRes: integer; { Vertical resolution of printer in dpi.}
iHRes: integer; { Horizontal resolution of printer in dpi.}
rPage: Rect; { Page (printable) rectangle - device coordinates.}
end;

```

TPPrInfo = ^TPrInfo;

FIG 4 - THE TPrint RECORD

## The Printing Graphics Port

**PrOpenDoc**, which opens a printing graphics port, returns a pointer to a **TPrPort** record. The **TPrPort** record, which defines a printing graphics port, is as follows:

```

type
TPrPort = record
gPort: GrafPort; { Printer's graphics port record.}
gProcs: QDProcs; { Procedures for printing in the graphics port.}
...             { More fields for internal use.}
end;

TPPrPort = ^TPrPort;
struct TPrPort

```

## Field Descriptions

---

<code>gPort</code>	A graphics port record, which may be either a <code>CGrafPort</code> or a <code>GrafPort</code> record, depending on whether the current printer supports colour and greyscale, and whether Color QuickDraw is available on the computer. <sup>2</sup>
<code>gProcs</code>	<p>A <code>QDProcs</code> record, which contains pointers to routines which the printer driver may have designated to take the place of QuickDraw routines.</p> <p>You print text and graphics by drawing into the 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 this field.</p>

## Print Status Dialog Boxes and Idle Procedure

---

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 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 and that the process may be aborted by pressing Command-period.

A user should always be able to cancel printing by pressing Command-period. To determine whether the user has cancelled printing, the printer driver periodically runs an **idle procedure**.

The `TPrJob` record contains a pointer to an idle procedure in its `pIdleProc` field (see Fig 4). If this field contains the value `NULL`, 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. Note, however, that 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.

To handle update information in your status dialog box during the printing operation, you should install your own idle procedure in the `pIdleProc` field of the `TPrJob` record. 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.

If you do not provide your own idle procedure, you can determine whether the user has cancelled printing by calling `PrError` to check for the `iPrAbort` error code after each call to a Printing Manager routine.

## Printing a Document - The Printing Loop

---

That part of your application's code which handles printing is referred to as the **printing loop**. A printing loop calls all the Printing Manager routines necessary to print a document, checking for printing errors at every step. In general, the printing loop should perform the following tasks:

- **Unload Unused Code Segments.** Unused code segments<sup>3</sup> should be unloaded to ensure that the maximum possible memory is available for printing.
- **Open the Printing Manager and Current Printer Driver.** Use `PrOpen` to initialise the Printing Manager and to open the printer driver for the current printer (that is, the printer the user last selected in the Chooser).

---

<sup>2</sup>If you need to determine the type of graphics port, you can check the high bit of the `rowBytes` field. If this bit is set, the printing graphics port is based on a `CGrafPort` record.

<sup>3</sup>See Chapter 21 — Miscellany.

- **Create or Validate a TPrint Record.** Use `NewHandle` to allocate storage for a `TPrint` record, and then initialise that `TPrint` record using `PrintDefault`. Alternatively, if you are using an existing `TPrint` record, use `PrValidate` to check that the record is compatible with the current printer and its driver.
- **Display the Job Dialog Box.** Use `PrJobDialog` to display the job dialog box<sup>4</sup> and to handle all user interaction in the standard dialog items until the user clicks the Print or Cancel button. Your application should print the document in the active window if the user clicks the Print button in the job dialog box.
- **Determine the Number of Copies and Number of Pages to Print.** Determine the number of copies to print, and the number of pages required to print the requested range of pages, by examining the fields of the `TPrint` record. (Note that, 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.)
- **Display a Status Dialog Box (Optional).** If required, display a printing status dialog box indicating to the user the status of the current printing operation.
- **Install an Idle Procedure (Optional).** If a status dialog box is used, install an idle procedure in the `pIdleProc` field of the `TPrJob` record to update information in the status dialog box and to check whether the user wants to cancel the printing operation.
- **Print the Requested Range of Pages.** Print the requested range of pages for each requested copy as follows:
  - **Open a Printing Graphics Port.** Call `PrOpenDoc` 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` (maximum pages in a spool file).<sup>5</sup>
  - **Open a Page for Printing.** Call `PrOpenPage` to set up the printing graphics port for the page. (`PrOpenPage` initialises the fields of the graphics port, and must be called for every page to be printed.)
  - **Draw in the Printing Graphics Port.** Use appropriate `QuickDraw` routines to draw into the printing graphics port.
  - **Close the Page.** When your application has finished drawing into the page, close the page using `PrClosePage`.
  - **Close the Printing Graphics Port.** Call `PrCloseDoc` to close the printing graphics port and begin printing the requested range of pages
  - **Check for Deferred Printing.** Check whether the printer driver is using deferred printing and, if so, call `PrPicFile` to send the spool file to the printer. (The `bjDocLoop` field of the `TPrJob` record is set to `bDraftLoop` (0) for draft and `bSpoolLoop` (1) for deferred printing.)
- **Close the Printing Manager.** The printing loop should then close the Printing Manager using `PrClose`. `PrClose` releases the Printing Manager dialog and other resources, but it leaves the printer driver open. (The printer driver may be closed using `PrDrvrclose`.)

## Creating and Validating the TPrint Record

The following example shows how to create a `TPrint` record. Note that `PrintDefault` is called to initialise the fields of the `TPrint` record according to the current printer's default values. (The default values are stored in the printer driver's resource file.)

<sup>4</sup>The `PrDlgMain` function is used to display a customized job dialog box.

<sup>5</sup>The value represented by `iPFMaxPgs` is 128.



```

tPrintHdl: THPrint;
...

tPrintHdl := THPrint(NewHandleClear(sizeof(TPrint)));
if (tPrintHdl <> NIL) then
begin
  PrintDefault(tPrintHdl);    { Sets appropriate default values for current driver.}
  printError := PrError
  if (printError <> noErr) then
    DoPrintError(printError);
  end
else
  ; { Handle error.}

```

You can also use an existing TPrint record (for example, one saved with a document). The following example application-defined function reads a record that the application has saved with a document as a resource of type 'SPRC'. Note that PrValidate is called to make sure that the TPrint record is valid for the current version of the Printing Manager and for the current printer driver.

```

function DoGetPrintRecord(refNum: longint; tPrintHdl: THPrint;
                          var prRecChanged: Boolean): OSErr;

var
  saveResFile: longint;
  result: OSErr;

begin
  saveResFile := CurResFile;
  UseResFile(refNum);

  tPrintHdl := THPrint(Get1Resource('SPRC', kDocPrintRec));
  if (tPrintHdl <> nil) then
  begin
    DetachResource(Handle(tPrintHdl));
    prRecChanged := PrValidate(tPrintHdl); { Check validity of TPrint record.}
    UseResFile(saveResFile);
    DoGetPrintRecord := OSErr(PrError);
  end
  else begin
    UseResFile(saveResFile);
    DoGetPrintRecord := kNilHandlePrintErr;
  end;
end;
{of function DoGetPrintRecord}

```

## Drawing in the Graphics Port

---

Observe the following general rules when drawing in the printing graphics port:

- Do not depend on values in the printing graphics port remaining identical from page to page. With each new page, you generally get re-initialised font information and other characteristics for the printing graphics port.
- Do not make calls which do not do anything on the printer. For example, QuickDraw erase routines are quite time-consuming and normally are not needed on the printer. Paper does not need to be erased the way the screen does.
- Do not use clipping to select text to be printed. There are a number of subtle differences between the way text appears on the screen and the way it appears on the printer, and you cannot count on knowing the exact dimensions of the rectangle occupied by the text.
- Do not use fixed-width fonts to align columns. Explicitly move the pen to where you want it.
- Do not use the outline font to create white text on a black background.
- Avoid changing fonts frequently.

Note that, because of the way rectangle intersections are determined, you slow printing substantially if your clipping region falls outside the rectangle given by the `rPage` field of the `TPrInfo` record.

## Handling Printing Errors

---

The Printing Manager must necessarily bear the heavy burden of maintaining backward compatibility with early Apple printer models and of maintaining compatibility with over a hundred existing printer drivers. For this reason, you must be especially wary of, and defensive about, possible error conditions when using Printing Manager routines and data structures.

`PrError` returns the result of the last Printing Manager function call. `PrError` returns `noErr` if no error occurred.

If you determine that an error has occurred after the completion of a printing routine, stop printing and 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`.

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

- If you display a dialog box in the middle of a printing loop, it could cause errors that might terminate an otherwise normal printing operation.
- 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 that it should cancel printing.

## An Example Printing Loop

---

The following is an example of a printing loop:

```
procedure PrintLoop(docToPrint : DocumentRecordHdl; displayJobDialog : boolean);

var
  oldPort : GrafPtr;
  numberOfPages, numberOfCopies : longint;
  userClickedOK : boolean;
  firstPage, lastPage, copy, page : longint;
  tprStatus : TPrStatus;
  printError : longint;

begin
  GetPort(oldPort);
  DoUnloadSegments;

  PrOpen;
  if (OSErr(PrError) = noErr) then
    begin
      gPrintResFile := CurResFile;
      gTPrintHdl := docToPrint^.docPrintRecordHdl;
      changed := PrValidate(gTPrintHdl);

      if (OSErr(PrError) = noErr) then
        begin
          numberOfPages := DoCalculateNumberOfPages(gTPrintHdl^^.prInfo.rPage);

          if (displayJobDialog) then
            userClickedOK := PrJobDialog(gTPrintHdl)
          else
            userClickedOK := DoJobMerge(gTPrintHdl);

          if (userClickedOK) then
            begin
              numberOfCopies := gTPrintHdl^.prJob.iCopies;
```

```

firstPage := gTPrintHdl ^^ . prJob. iFstPage;
lastPage := gTPrintHdl ^^ . prJob. iLstPage;

gTPrintHdl ^^ . prJob. iFstPage := 1;
gTPrintHdl ^^ . prJob. iLstPage := iPrPgMax;

if (numberOfPages < lastPage) then
    lastPage := numberOfPages;

DoActivateFrontWindow(false, oldPort);
gPrintStatusDlg := GetNewDialog(rPrintStatus, NIL, WindowPtr(-1)); { Optional }
DoDialogBoxItems(docToPrint); { Optional }
ShowWindow(gPrintStatusDlg); { Optional }
gTPrintHdl ^^ . prJob. pIdleProc := @DoPrintIdle; { Optional }

for copy := 1 to numberOfCopies do
    begin
        UseResFile(gPrintResFile);

        for page := firstPage to lastPage do
            begin
                if (((page - firstPage) mod iPFMaxPgs) = 0)
                    begin
                        if (page <> firstPage) then
                            begin
                                PrCloseDoc(gPrintPortPtr);

                                if ((gTPrintHdl ^^ . prJob. bJDocLoop = bSpoolLoop) and (PrError = 0))
                                    then PrPicFile(gTPrintHdl, NIL, NIL, NIL, tprStatus);
                                end;
                                gPrintPortPtr := PrOpenDoc(gTPrintHdl, NIL, NIL);
                                end;
                            if (OSErr(PrError) = noErr) then
                                begin
                                    PrOpenPage(gPrintPortPtr, NIL);
                                    if (OSErr(PrError) = noErr) then
                                        DoDrawPrintPage(gTPrintHdl ^^ . prInfo. rPage, docToPrint,
                                                            GrafPtr(gPrintPortPtr), page);
                                    PrClosePage(gPrintPortPtr);
                                    end;
                                end;
                            end;

                        PrCloseDoc(gPrintPortPtr);

                        if ((gTPrintHdl ^^ . prJob. bJDocLoop = bSpoolLoop) and (OSErr(PrError) = noErr))
                            then PrPicFile(gTPrintHdl, NIL, NIL, NIL, tprStatus);
                        end;
                    end;
                end;
            end;

        printError := PrError;

        PrClose;

        if (OSErr(printError) <> noErr) then
            DoPrintError(printError);

        DisposeDialog(gPrintStatusDlg);
        SetPort(oldPort);
        DoActivateFrontWindow(true, oldPort);
    end;
    {of procedure PrintLoop}

```

## **Preliminaries**

**PrintLoop** begins by saving a pointer to the current graphics port and swapping out code segments not required during printing. It then opens the Printing Manager, together with the current printer driver and its resource file, by calling **PrOpen**. Note that the current resource file is now the printer driver's resource file. Assuming no error, the current resource file is saved so that, if **printLoop**'s idle procedure changes the resource chain in any way, it can restore the current resource file before returning.

`PrValidate` is then used to change any values in the `TPrint` record associated with the document to match those specified by the current driver. (`PrValidate`, rather than `PrDefault`, is used so as to preserve any values the user may have previously set through the style dialog box.)

## **Calculate Number of Pages**

---

The application-defined function `DoCalculateNumberOfPages` is called to divide the data in the file into sections that fit within the printable page rectangle stored in the `rPage` field of the `TPrInfo` record and, by so doing, to determine the number of pages required to print the document.

## **Display Job Dialog Box or Perform Job Merge**

---

If the calling routine so specifies, the job dialog box is then displayed. (If the user prints multiple documents at once, the calling routine sets the `displayJobDialog` parameter to `true` for the first document and `false` for the rest. This allows the user to specify the values in the job dialog box only once when printing multiple documents. It also facilitates the printing of documents in the background (for example, as the result of responding to the required Apple event `Print Documents`) without requiring the application to display the job dialog box.)

If `displayJobDialog` was set to `false` by the calling routine, the application-defined function `DoJobMerge` would, amongst other things, use `PrJobMerge` to copy data from the first print record to the print record for the document about to be printed.

## **Get First Page, Last Page, and Number of Copies**

---

If `true` is returned by either the call to `PrJobDialog` (that is, the user clicked the `Print (OK)` button) or the call to `DoJobMerge` (that is, there is another document to print), the number of copies, first page and last page are retrieved from the relevant fields of the `TPrJob` record. Since the only information which should be preserved between separate printings of the same document is that obtained via the style dialog box, the fields of the `TPrJob` record which store the first and last page numbers are then set back to 1 and `iPrPgMax` (9999) respectively.

If the last page number specified by the user exceeds the total number of pages in the document, the variable holding the last page value is set to the actual number of pages.

## **Display a "Print Status" Dialog Box and Install an Idle Procedure (Optional)**

---

Before sending the pages off to be printed, a "print status" dialog is displayed to inform the user of the current status of the printing operation. If the dialog provides a button, or reports on the progress of the printing operation, an idle procedure must be installed to handle events in the dialog. The printer driver calls the idle procedure periodically during the printing process.

The following is an example of an application-defined idle procedure which assumes the use of a modal dialog box to display printing status information:

```
procedure DoPrintIdle;

var
  oldPort : GrafPtr;
  event : EventRecord;
  gotEvent : boolean;
  itemHit : longint;
  handled, cancelled : boolean;

begin
  GetPort(oldPort);
  SetPort(gPrintStatusDlg);

  gotEvent := WaitNextEvent(everyEvent, event, 15, NIL);
```

```

if (gotEvent) then
begin

    { doHandleEvent should handle update and activate events. This also enables }
    { background applications to receive update events while the "print status" modal }
    { dialog is open. }

    handled := DoHandleEvent(gPrintStatusDlg, event, itemHit);

    { doDidUserCancel should scan for Command-period key-down events (see Chapter 22 - }
    { Miscellany) and also for mouse-down events indicating that the user clicked the }
    { Stop Printing button. }

    cancelled := DoDidUserCancel; { Scan for Command-period or Cancel button click. }
    if (cancelled) then
        itemHit := kStopButton;

    { To handle hits in the "print status" dialog, doHandleHitsInStatusBox should }
    { simply check the item number passed to it. For the Stop Printing button, it }
    { should call PrSetError, specifying the error code iPrAbort. For hits in other }
    { items, it should set the cursor to a spinning wristwatch cursor. }

    handled := DoHandleHitsInStatusBox(itemHit);
end;

{ doUpdateStatus should update those items in "print status" dialog box that report }
{ printing status the user. }

DoUpdateStatusInformation(cancelled); { Update items in status dialog box. }

SetPort(oldPort);
end;
{of procedure DoPrintIdle}

```

The following guidelines should be followed when writing your own idle procedure:

- If you draw anything within the idle procedure, save the printing graphics port upon entry to the idle procedure and restore it upon exit, as shown in the example.
- If your idle procedure changes the resource chain<sup>6</sup>, save the reference number of the printer driver's resource file by calling `CurResFile` at the beginning of your idle procedure. Upon exit, restore the resource chain using `UseResFile`.
- Avoid calling `PrError` within the idle procedure.

## Copies Loop

Before beginning the actual printing process, `PrintLoop` displays its own status dialog box and installs its own idle procedure. A loop, which will cycle once for each of the specified number of copies, is then entered. The current resource file is restored to the printer driver's resource file at the top of this loop.

## Pages Loop

A nested loop is then entered for the printing of each page. The maximum number of pages that can be printed at a time is represented by the constant `iPFMaxPgs` (128). If 128 pages have been printed, the printing graphics port is closed by a call to `PrCloseDoc` and, if the printer driver is using deferred printing, `PrPicFile` is called to send the spool file to the printer. If this is either the first page of all or the first page after the first 128 have been printed, `PrOpenDoc` is called to initialise a printing graphics port and make it the current port.

For each page, `PrOpenPage` is called to initialise the printing graphics port, the application-defined routine `DoDrawPrintPage` is called to draw the page in the printing graphics port, and `PrClosePage` is called to wrap up printing of the current page. (Note that the parameters taken by `DoDrawPrintPage` are the size of the page rectangle, the document containing the page to print, the printing graphics port in

<sup>6</sup>See Chapter 15 — More on Resources.

which to draw, and the page number. This allows the application to use the same code to print a page as it uses to draw the same page on the screen.)

## **Exit From the Copies Loop**

---

When all pages have been printed, `PrCloseDoc` is called to close the printing graphics port. If the printer driver is using deferred printing, `PrPicFile` is called to send the spool file to the printer. Finally, `PrClose` is called to release memory associated with the Printing Manager (except the printer driver). It then remains to dispose of the status dialog, reset the current graphics port and activate the application's front window.

## **Getting and Setting Printer Information**

---

By using `PrGeneral` you can determine the resolution of the printer, set the printer resolution, ascertain if the user has set landscape printing, and force enhanced draft-quality printing.

To achieve these ends, you use `PrGeneral` with one of five opcodes: `getRslDataOp`, `setRslOp`, `getRotnOp`, `draftBitsOp`, or `noDraftBitsOp`. These opcodes have data structures associated with them. When you call `PrGeneral`, `PrGeneral`, in turn, calls the current printer driver to get or set the desired information.

## **Checking Whether the Current Printer Driver Supports PrGeneral**

---

Note that not all printer drivers support all of the features provided by `PrGeneral`. The following example application-defined function checks whether the current printer driver supports `PrGeneral`.

```
function DoIsPrGeneralThere : boolean;

var
  getRotRec : TGetRotnBlk;
  printError : OSErr;

begin
  printError := 0;
  getRotRec.iOpCode := getRotnOp; { Set opcode used to determine if landscape chosen. }
  getRotRec.hPrint := gTPrintHdl; { TPrint record this operation applies to. }

  PrGeneral (Ptr(@getRotRec));

  printError := OSErr(PrError);
  PrSetError(noErr);

  if (printError = resNotFound) then
    DoIsPrGeneralThere := false
  else
    DoIsPrGeneralThere := true;
end;
```

## **Using PrGeneral to Determine Page Orientation**

---

The principal use of `PrGeneral` is probably to determine page orientation. This can be useful where, for example, an image will only fit on the page in landscape orientation, the user has not selected landscape, and you want your application to remind the user to select landscape before printing so as to avoid a clipped printed image. The following is an example application-defined function which returns a value indicating whether the user has selected landscape orientation:

```
function DoGetPageOrientation : integer;

var
  TGetRotnBlk getRotRec;

begin
  if (doIsPrGeneralThere)
  then begin
    getRotRec.iOpCode = getRotnOp;
    getRotRec.hPrint = gTPrintHdl;
```

```

PrGeneral((Ptr) &getRotRec);
if ((getRotRec.iError = noErr) and (OSErr(PrError) = noErr) and getRotRec.fLandscape)
    then DoGetPageOrientation := kInLandscapeOrientation
    else
        DoGetPageOrientation := kInPortraitOrientation;
end
else DoGetPageOrientation := kPrGeneralAbsent;
end;

```

## Error Handling

---

When using `PrError` and `PrGeneral`, be prepared to receive the errors `noSuchRsl` (printer does not support the requested resolution), `opNotImpl` (printer does not support the `PrGeneral` opcode selected) and `resNotFound` (current printer driver does not support `PrGeneral`). If you receive a `resNotFound` result code, clear the error by calling `PrSetError` with a value of `noErr`.

## Text on the Screen and the Printed Page

---

At the application level, printing on the Macintosh computer is not fundamentally different from drawing on the screen. That said, printing text poses special challenges.

A common complication results from the difference in resolution and pixel size between screen and printer. QuickDraw measurements are theoretically in terms of **points**, which are nominally equivalent to screen pixels. High resolution printers have very much smaller pixels, although printer drivers are expected to take this into account so that the same QuickDraw calls will produce text lines of the same width on the screen and on the printer. Nevertheless, this higher resolution, and the fact that printers can use different fonts from those used for screen display, can result in some loss of fidelity from the screen to the printed page. In this regard, the following is relevant:

- QuickDraw places text glyphs<sup>7</sup> on the screen at screen pixel intervals, whereas a printer can provide much finer placements on the printed page. This situation presents a choice between optimising the appearance of text on the screen or on the printed page. In effect, that choice is whether to specify **fractional glyph widths** or integer glyph widths.

Fractional glyph widths are measurements of a glyph's width which can include fractions of a pixel. Using fractional glyph widths improves the appearance of printed text because it makes it possible for the printer, with its very high resolution, to print with better spacing. However, because screen glyphs are made up of whole pixels, QuickDraw cannot draw a fractional glyph on the screen, so it rounds off the fractional parts. This results in some degradation in the appearance of the text, in terms of character spacing, on the screen.

The alternative (integer glyph widths) gives more pleasing screen results because the characters are drawn with regular pixel spacing, but this may possibly be at the price of a printed page which is typographically unacceptable.

The Font Manager routine `SetFractEnable` is used to turn fractional glyph widths on and off. `SetFractEnable` affects routines which draw text and which calculate text and character widths.

- Printer drivers attempt to reproduce faithfully the text formatting as drawn by QuickDraw on the screen, including keeping the same intended character spacing, line breaks and page breaks. However, because printers can have resident fonts that are different from the fonts that QuickDraw uses, because the drivers may handle text layout somewhat differently than QuickDraw, and because font metrics do not always scale linearly, fidelity may not always be achieved. Typically, identical line breaks and page breaks can be maintained, but character spacing can be noticeably different.

---

<sup>7</sup>A glyph is the visual representation of a character. See Chapter 17 — Text and TextEdit.

## Altering the Style or Job Dialog Box

---

You may want to add additional options to the style and job dialog boxes so that the user can further customise the printing process. For example, you might want to add a "skip blank pages" checkbox to a job dialog box. You can customise a style or job dialog box by taking the following steps:

- Use `PrOpen` to open the Printing Manager.
- Use `PrStlInit` or `PrJobInit` to initialise a `TprDlg` record. (This record contains the information needed to set up the style or job dialog box.)
- Define an initialisation routine that appends items to the printer driver's style or job dialog box. The initialisation routine should:
  - Use `AppendDITL` to add items to the dialog box whose `TprDlg` record you have initialised 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 does not handle in a modal dialog box, and one in the `pItemProc` field for handling events in the items added to the dialog box.
  - Return a pointer to the `TprDlg` record.
- Pass the address of your initialisation routine to `PrDlgMain` to display the dialog box.
- Respond to the dialog box as appropriate.
- Use `PrClose` when you are finished using the Printing Manager.

## Printing From the Finder

---

Users generally print documents that are open on the screen one at a time while the application that created the document is running. However, users can also print one or more documents from the Finder by selecting the documents and choosing Print... from the Finder's File menu. This causes the Finder to launch the application and pass it a required Apple event (the Print Documents event) indicating the documents to be printed. In response to a Print Documents event, your application should:

- Open windows for the documents only if your application can interact with the user (see Chapter 8 - Required Apple Events.)
- Use saved or default style settings instead of displaying the style dialog box.
- Display the job dialog box once only, and use `PrJobMerge` to apply the information specified by the user to all of the selected documents. (Note that `PrJobMerge` preserves the fields of the `TPrint` record that are specific to each document, that is, the fields that are set through the style dialog box.)
- Remain open unless and until the Finder sends it a Quit Application event.

## Main Printing Manager Constants, Data Types and Routines

---

### Constants

---

<code>iPFMaxPgs</code>	= 128	Maximum pages in spool file.
<code>iPrPgFract</code>	= 120	Page scale factor.
<code>iPrPgFst</code>	= 1	Page range constant - first page.



iPrPgMax	= 9999	Page range constant - last page.
bDraftLoop	= 0	Draft-quality printing.
bSpoolLoop	= 1	Deferred printing.

## PrGeneral Opcodes

getRslDataOp	= 4	Get resolutions for 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.
NoSuchRsl	= 1	Resolution not supported.

## Data Types

---

### Print Record

```
TPrint = record
  iPrVersion: integer;           { (Reserved)}
  prInfo:      TPrInfo;         { PrInfo data associated with the current style.}
  rPaper:      Rect;            { Paper rectangle (offset from rPage).}
  prStl:       TPrStl;          { This print request's style.}
  prInfoPT:    TPrInfo;         { (Reserved)}
  prXInfo:     TPrXInfo;        { (Reserved)}
  prJob:       TPrJob;          { Print Job request.}
  case integer of
    0: (
      printX:    array [1..19] of integer; { (Reserved)}
    );
    1: (
      prFlag1:   TPrFlag1;
      iZoomMin:  integer;
      iZoomMax:  integer;
      hDocName:  StringHandle;
    );
  end;
TPPrint = ^TPrint;
THPrint = ^TPPrint;
```

### Printer Information Record

```
TPrInfo = record
  iDev:      integer;           { (Reserved)}
  iVRes:     integer;           { Vertical resolution of printer in dpi.}
  iHRes:     integer;           { Horizontal resolution of printer in dpi.}
  rPage:     Rect;             { Page (printable) rectangle in device coordinates.}
end;
TPPrInfo = ^TPrInfo;
```

### Print Job Record

```
TPrJob = record
  iFstPage:   integer;          { First page of page range.}
  iLstPage:   integer;          { Last page of page range.}
  iCopies:    integer;          { Number of copies.}
  bJDocLoop:  SInt8;            { Printing method - draft or deferred.}
  fFromUsr:   boolean;          { (Reserved)}
  pIdleProc:  PrIdleUPP;        { Pointer to an idle procedure.}
  pFileName:  StringPtr;        { Spool file name: NIL for default.}
  iFileVol:   integer;          { Spool file volume, set to 0 initially}
  bFileVers:  SInt8;            { Spool file version, set to 0 initially}
  bJobX:      SInt8;            { (Reserved)}
end;
TPPrJob = ^TPrJob;
```

### Printing Style Record

```
TPrStl = record
  wDev:      integer;           { Device number of printer.}
  iPageV:    integer;           { (Reserved)}
  iPageH:    integer;           { (Reserved)}
```

```

bPort:      SInt8;          { (Reserved)}
feed:       TFeed;         { Feed type.}
end;

```

```
TPPrSt1 = ^TPrSt1;
```

## Printing Graphics Port Record

```

TPrPort = record
  gPort:      GrafPort;     { Graphics port for printing.}
  gProcs:     QDDProcs;     { Procedures for printing in graphics port.}
  lGParam1:   longint;      { 16 bytes for private parameter storage.}
  lGParam2:   longint;      { (Reserved)}
  lGParam3:   longint;      { (Reserved)}
  lGParam4:   longint;      { (Reserved)}
  fOurPtr:    boolean;      { (Reserved)}
  fOurBits:   boolean;      { (Reserved)}
end;

```

```
TPPrPort = ^TPrPort;
```

## Printing Status Record

```

TPrStatus = record
  iTotPages:  integer;      { Total pages in Print File.}
  iCurPage:  integer;      { Current page number}
  iTotCopies: integer;      { Current copies requested}
  iCurCopy:  integer;      { Current copy number}
  iTotBands:  integer;      { Total bands per page.}
  iCurBand:  integer;      { Current band number}
  fPgDirty:   boolean;      { True if current page has been written to.}
  fImaging:   boolean;      { Set while in band's DrawPic call.}
  hPrint:     THPrint;      { Handle to the active printer record}
  pPrPort:    TPPrPort;     { Pointer to the active printing graphics port.}
  hPic:       PicHandle;     { Handle to the active picture}
end;

```

## Print Dialog Box Record

```

TPrDlg = record
  Dlg:      DialogRecord; { The dialog window}
  pFiltrProc: ModalFilterUPP; { The filter proc.}
  pItemProc: PItemUPP;    { The item evaluating proc.}
  hPrintUsr: THPrint;      { The user's print record.}
  fDoIt:     boolean;      { true means user clicked OK.}
  fDone:     boolean;      { true means user clicked OK or Cancel}
  lUser1:    longint;      { Four longs for apps to hang global data.}
  lUser2:    longint;      { Plus more stuff needed by the particular }
  lUser3:    longint;      { printing dialog.}
  lUser4:    longint;
end;

```

```
TPPrDlg = ^TPrDlg;
```

```
PDlgInitUPP = UniversalProcPtr;
```

## Page Orientation Record

```

TGetRotnBlk = 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:    SInt8;        { (Reserved)}
end;

```

```
TPRect = ^Rect;
```

```
PrIdleProcPtr = UniversalProcPtr;
```

```
PItemUPP = UniversalProcPtr;
```

## Routines

---

### Opening and Closing the Printing Manager

```
procedure PrOpen;
procedure PrClose;
```

### Initialising and Validating TPrint Records

```
procedure PrintDefault(hPrint: THPrint);
function PrValidate(hPrint: THPrint): boolean;
```

### Displaying and Customising Print Dialog Boxes

```
function PrStlDialog(hPrint: THPrint): boolean;
function PrJobDialog(hPrint: THPrint): boolean;
function PrStlInit(hPrint: THPrint): TPrDlgRef;
function PrJobInit(hPrint: THPrint): TPrDlgRef;
procedure PrJobMerge(hPrintSrc: THPrint; hPrintDst: THPrint);
function PrDlgMain(hPrint: THPrint; pDlgInit: PDlgInitUPP): boolean;
```

### Printing a Document

```
function PrOpenDoc(hPrint: THPrint; pPrPort: TPrPort; pIOBuf: Ptr): TPrPort;
procedure PrCloseDoc(pPrPort: TPrPort);
procedure PrOpenPage(pPrPort: TPrPort; pPageFrame: TRect);
procedure PrClosePage(pPrPort: TPrPort);
procedure PrPicFile(hPrint: THPrint; pPrPort: TPrPort; pIOBuf: Ptr; pDevBuf: Ptr;
VAR prStatus: TPrStatus);
```

### Optimising Printing

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

### Handling Printing Errors

```
function PrError: integer;
procedure PrSetError(iErr: integer);
```

## Demonstration Program

---

```
1 { #####
2 // PrintingPascal.p
3 // #####
4 //
5 // This program:
6 //
7 // • Opens a window in which the contents of the main fields in the TPrint, TPrJob,
8 //   TPrStl and TPrInfo records are displayed.
9 //
10 // • Allows the user to note changes in these fields after invoking the style dialog
11 //   and job dialog boxes.
12 //
13 // • Allows the user to print a simulated document.
14 //
15 // • Quits when the user chooses Quit or clicks the window's close box.
16 //
17 // The program utilises the following resources:
18 //
19 // • 'MBAR' resource and associated 'MENU' resources (preload, non-purgeable).
20 //
21 // • A 'WIND' resource (purgeable).
22 //
23 // • An 'ALRT' resource and associated 'DITL' resource for an alert which reports
24 //   printing errors (purgeable).
25 //
26 // • A 'TEXT' resource (non-purgeable) used for printing.
27 //
28 // • A 'PICT' resource (non-purgeable) used for printing.
29 //
30 // ##### }
```

```

31
32 program PrintingPascal(input, output);
33
34 { ..... include the following Universal Interfaces }
35
36 uses
37
38     Windows, Fonts, Menus, TextEdit, Quickdraw, Dialogs, QuickdrawText, Processes, Types,
39     Memory, Events, TextUtils, ToolUtils, OSUtils, Devices, Printing, Resources,
40     SegLoad, Errors;
41
42 { ..... define the following constants }
43
44 const
45
46     mApple = 128;
47     mFile = 129;
48     iQuit = 11;
49     iPageSetup = 8;
50     iPrint = 9;
51     rMenubar = 128;
52     rWindow = 128;
53     rText = 128;
54     rPicture = 128;
55     rPrintAlert = 128;
56     kMargin = 90;
57     kMaxLong = $7FFFFFFF;
58
59 { ..... global variables }
60
61 var
62
63     gTPrintHdl : THPrint;
64     gWindowPtr : WindowPtr;
65     gDone : boolean;
66     gPrintRecordInitd : boolean;
67     gInhibitPrintRecordsInfo : boolean;
68     gEditRecHdl : TEHandle;
69     gTextHdl : Handle;
70     gPictureHdl : PicHandle;
71     menubarHdl : Handle;
72     menuHdl : MenuHandle;
73     eventRec : EventRecord;
74     gotEvent : boolean;
75
76 { ##### DoInitManagers }
77
78 procedure DoInitManagers;
79
80     begin
81         MaxApplZone;
82         MoreMasters;
83
84         InitGraf(@qd.thePort);
85         InitFonts;
86         InitWindows;
87         InitMenus;
88         TEInit;
89         InitDialogs(nil);
90
91         InitCursor;
92         FlushEvents(everyEvent, 0);
93     end;
94     {of procedure DoInitManagers}
95
96 { ##### DoPrintError }
97
98 procedure DoPrintError(printError : longint; fatal : boolean);
99
100     var
101         errorNumberString : string;
102         ignored : OSErr;
103
104     begin
105         NumToString(printError, errorNumberString);
106         ParamText(errorNumberString, '', '', '');
107         if (fatal) then

```

```

108     begin
109         ignored := StopAlert(rPrintAlert, nil);
110         ExitToShell;
111     end
112 else
113     ignored := CautionAlert(rPrintAlert, nil);
114 end;
115 {of procedure DoPrintError}
116
117 { ##### DoIsPrGeneralThere }
118
119 function DoIsPrGeneralThere : boolean;
120
121     var
122         getRotRec : TGetRotnBlk;
123         printError : OSerr;
124
125     begin
126         printError := 0;
127         getRotRec.iOpCode := getRotnOp;
128         getRotRec.hPrint := gTPrintHdl;
129
130         PrGeneral(Ptr(@getRotRec));
131
132         printError := PrError;
133         PrSetError(noErr);
134
135         if (printError = resNotFound) then
136             DoIsPrGeneralThere := false
137         else
138             DoIsPrGeneralThere := true;
139         end;
140     {of function DoIsPrGeneralThere}
141
142 { ##### DoGetPageOrientation }
143
144 function DoGetPageOrientation : longint;
145
146     var
147         getRotRec : TGetRotnBlk;
148
149     begin
150         if (DoIsPrGeneralThere) then
151             begin
152                 getRotRec.iOpCode := getRotnOp;
153                 getRotRec.hPrint := gTPrintHdl;
154                 PrGeneral(Ptr(@getRotRec));
155                 if ((getRotRec.iError = noErr) and (PrError = noErr) and (getRotRec.fLandscape))
156                     then DoGetPageOrientation := 1
157                     else DoGetPageOrientation := 2;
158             end
159         else
160             DoGetPageOrientation := 3;
161         end;
162     {of function DoGetPageOrientation}
163
164 { ##### DoDrawPageOrientation }
165
166 procedure DoDrawPageOrientation;
167
168     var
169         orientation : longint;
170
171     begin
172         MoveTo(20, 260);
173         DrawString('Orientation selected:');
174
175         orientation := DoGetPageOrientation;
176
177         MoveTo(190, 260);
178         if (orientation = 1) then
179             DrawString('Landscape')
180         else if (orientation = 2) then
181             DrawString('Portrait')
182         else
183             DrawString('(PrGeneral not supported by driver)');
184         end;

```

```

185     {of procedure DoDrawPageOrientation}
186
187 { ##### DoDrawRectStrings }
188
189 procedure DoDrawRectStrings(s1: string; x1, y1: integer; s2: string; x2, y2: integer;
190     s3: string);
191
192     begin
193     MoveTo(x1, y1);
194     DrawString(s1);
195     MoveTo(x2, y2);
196     DrawString(' ');
197     DrawString(s2);
198     DrawString(' ');
199     DrawString(s3);
200     DrawString(' ');
201     end;
202     {of procedure DoDrawRectStrings}
203
204 { ##### DoPrintRecordsInfo }
205
206 procedure DoPrintRecordsInfo;
207
208     var
209     s2, s3 : string;
210
211     begin
212     EraseRect(gWindowPtr^.portRect);
213
214     MoveTo(20, 25);
215     DrawString(' From TPrint, TPrInfo and TPrStl records: ');
216
217     NumToString(integer(gTPrintHdl^.rPaper.top), s2);
218     NumToString(integer(gTPrintHdl^.rPaper.left), s3);
219     DoDrawRectStrings(' Paper Rectangle (top, left): ', 20, 45, s2, 190, 45, s3);
220
221     NumToString(integer(gTPrintHdl^.rPaper.bottom), s2);
222     NumToString(integer(gTPrintHdl^.rPaper.right), s3);
223     DoDrawRectStrings(' Paper Rectangle (bottom, right): ', 20, 60, s2, 190, 60, s3);
224
225     NumToString(integer(gTPrintHdl^.prInfo.rPage.top), s2);
226     NumToString(integer(gTPrintHdl^.prInfo.rPage.left), s3);
227     DoDrawRectStrings(' Page Rectangle (top, left): ', 20, 75, s2, 190, 75, s3);
228
229     NumToString(integer(gTPrintHdl^.prInfo.rPage.bottom), s2);
230     NumToString(integer(gTPrintHdl^.prInfo.rPage.right), s3);
231     DoDrawRectStrings(' Page Rectangle (bottom, right): ', 20, 90, s2, 190, 90, s3);
232
233     MoveTo(20, 105);
234     DrawString(' Feed Type: ');
235     MoveTo(190, 105);
236     if (gTPrintHdl^.prStl.feed = 0) then
237         DrawString(' Cut sheet')
238     else if (gTPrintHdl^.prStl.feed = 1) then
239         DrawString(' Fanfold');
240
241     MoveTo(20, 120);
242     DrawString(' Vertical resolution: ');
243     NumToString(integer(gTPrintHdl^.prInfo.iVRes), s2);
244     MoveTo(190, 120);
245     DrawString(s2);
246
247     MoveTo(20, 135);
248     DrawString(' Horizontal resolution: ');
249     NumToString(integer(gTPrintHdl^.prInfo.iHRes), s2);
250     MoveTo(190, 135);
251     DrawString(s2);
252
253     MoveTo(20, 155);
254     DrawString(' From TPrJob Record: ');
255
256     MoveTo(20, 175);
257     DrawString(' First Page: ');
258     NumToString(integer(gTPrintHdl^.prJob.iFstPage), s2);
259     MoveTo(190, 175);
260     DrawString(s2);
261

```

```

262 MoveTo(20, 190);
263 DrawString(' Last Page: ');
264 NumToString(integer(gTPrintHdl ^^ . prJob . i LstPage), s2);
265 MoveTo(190, 190);
266 DrawString(s2);
267
268 MoveTo(20, 205);
269 DrawString(' Number of Copies: ');
270 NumToString(integer(gTPrintHdl ^^ . prJob . i Copies), s2);
271 MoveTo(190, 205);
272 DrawString(s2);
273
274 MoveTo(20, 225);
275 DrawString(' Note: Some printer drivers always set the iCopies field of the TPrJob');
276 MoveTo(20, 240);
277 DrawString(' record to 1 and handle multiple copies internally. ');
278 end;
279 {of procedure DoPrintRecordsInfo}
280
281 { ##### DoActivateWindow }
282
283 procedure DoActivateWindow;
284
285 begin
286 if ((FrontWindow = gWindowPtr) and gPrintRecordInitd
287 and not gInhibitPrintRecordsInfo) then
288 DoPrintRecordsInfo;
289 end;
290 {of procedure DoActivateWindow}
291
292 { ##### DoDrawPage }
293
294 procedure DoDrawPage(pageRect : Rect; pageNumber, numberOfPages : integer);
295
296 var
297
298 destRect, pictureRect : Rect;
299 heightDestRect, linesPerPage, numberOfLines, fontNum : SInt16;
300 pageEditRecHdl : TEHandle;
301 textHdl : Handle;
302 startOffset, endOffset : SInt32;
303 theString : Str255;
304
305 begin
306 SetRect(destRect, pageRect.left + kMargin, pageRect.top + (trunc(kMargin * 1.5)),
307 pageRect.right - kMargin, pageRect.bottom - (trunc(kMargin * 1.5)));
308
309 heightDestRect := destRect.bottom - destRect.top;
310 linesPerPage := trunc(heightDestRect / gEditRecHdl ^^ . lineHeight);
311 numberOfLines := gEditRecHdl ^^ . nLines;
312
313 GetFNum(' Geneva', fontNum);
314 TextFont(fontNum);
315 TextSize(10);
316
317 pageEditRecHdl := TENew(destRect, destRect);
318 textHdl := gEditRecHdl ^^ . hText;
319
320 startOffset := gEditRecHdl ^^ . lineStarts[(pageNumber - 1) * linesPerPage];
321 if (pageNumber = numberOfPages) then
322 endOffset := gEditRecHdl ^^ . lineStarts[numberOfLines]
323 else
324 endOffset := gEditRecHdl ^^ . lineStarts[pageNumber * linesPerPage];
325
326 HLock(textHdl);
327 TEInsert(Ptr(SInt32(textHdl ^) + startOffset), endOffset - startOffset, pageEditRecHdl);
328 HUnlock(textHdl);
329
330 if (pageNumber = 1) then
331 begin
332 SetRect(pictureRect, destRect.left, destRect.top,
333 destRect.left + (gPictureHdl ^^ . picFrame.right - gPictureHdl ^^ . picFrame.left),
334 destRect.top + (gPictureHdl ^^ . picFrame.bottom - gPictureHdl ^^ . picFrame.top));
335 DrawPicture(gPictureHdl, pictureRect);
336 end;
337
338 MoveTo(destRect.left, pageRect.bottom - 25);

```

```

339 NumToString(SInt32 (pageNumber), theString);
340 DrawString(theString);
341 end;
342 {of procedure DoDrawPrintPage}
343
344 { ##### DoCalcNumberOfPages }
345
346 function DoCalcNumberOfPages(pageRect : Rect) : integer;
347
348 var
349     destRect, pictureRect : Rect;
350     fontNum, heightDestRect, linesPerPage, numberOfPages : SInt16;
351
352 begin
353     EraseRect(gWindowPtr^.portRect);
354
355     SetRect(destRect, pageRect.left + kMargin, pageRect.top + (trunc(kMargin * 1.5)),
356             pageRect.right - kMargin, pageRect.bottom - (trunc(kMargin * 1.5)));
357     OffsetRect(destRect, - (kMargin - 5), - ((trunc(kMargin * 1.5)) - 5));
358
359     GetFNum(' Geneva', fontNum);
360     TextFont(fontNum);
361     TextSize(10);
362
363     gEditRecHdl := TENew(destRect, destRect);
364     TEInsert(gTextHdl^, GetHandleSize(gTextHdl), gEditRecHdl);
365
366     heightDestRect := destRect.bottom - destRect.top;
367     linesPerPage := trunc(heightDestRect / gEditRecHdl^^.lineHeight);
368     numberOfPages := trunc((gEditRecHdl^^.nLines / linesPerPage) + 1);
369
370     SetRect(pictureRect, destRect.left, destRect.top,
371             destRect.left + (gPictureHdl^^.picFrame.right - gPictureHdl^^.picFrame.left),
372             destRect.top + (gPictureHdl^^.picFrame.bottom - gPictureHdl^^.picFrame.top));
373     DrawPicture(gPictureHdl, pictureRect);
374
375     DoCalcNumberOfPages := numberOfPages;
376 end;
377 {of procedure DoCalcNumberOfPages}
378
379 { ##### DoCreatePrintRecord }
380
381 function DoCreatePrintRecord : OSErr;
382
383 var
384     printError : integer;
385
386 begin
387     gTPrintHdl := THPrint(NewHandleClear(sizeof(TPrint)));
388     if (gTPrintHdl <> nil) then
389     begin
390         PrintDefault(gTPrintHdl);
391         printError := PrError;
392         if (printError = noErr) then
393             gPrintRecordInitd := true;
394         DoCreatePrintRecord := printError;
395     end
396     else
397         ExitToShell;
398 end;
399 {of procedure DoCreatePrintRecord}
400
401 { ##### DoPrStyleDialog }
402
403 procedure DoPrStyleDialog;
404
405 var
406     printError : OSErr;
407     ignored : boolean;
408
409 begin
410     PrOpen;
411     printError := PrError;
412
413     if (printError = noErr) then
414     begin
415         if not (gPrintRecordInitd) then

```



```

416         begin
417             printError := DoCreatePrintRecord;
418             if (printError <> noErr) then
419                 DoPrintError(printError, true);
420             end;
421             ignored := PrStlDialog(gTPrintHdl);
422         end
423     else
424         DoPrintError(printError, false);
425     end;
426     PrClose;
427 end;
428 {of procedure DoPrStyleDialog}
429
430 { ##### PrintLoop ##### }
431
432 procedure PrintLoop;
433
434     var
435         oldPort : GrafPtr;
436         printError : integer;
437         numberOfPages, numberOfCopies : integer;
438         userClickedOK : Boolean;
439         firstPage, lastPage, copy, page : integer;
440         printPortPtr : TPrPort;
441         printStatus : TPrStatus;
442
443     begin
444         GetPort(oldPort);
445
446         PrOpen;
447         if (PrError = noErr) then
448             begin
449                 if not (gPrintRecordInited) then
450                     printError := DoCreatePrintRecord
451                 else
452                     printError := noErr;
453             end
454         if (printError = noErr) then
455             begin
456                 numberOfPages := DoCalcNumberOfPages(gTPrintHdl^^.prInfo.rPage);
457
458                 userClickedOK := PrJobDialog(gTPrintHdl);
459                 if (userClickedOK) then
460                     begin
461                         DoPrintRecordsInfo;
462                         DoDrawPageOrientation;
463                         gInhibitPrintRecordsInfo := true;
464
465                         numberOfCopies := gTPrintHdl^^.prJob.iCopies;
466                         firstPage := gTPrintHdl^^.prJob.iFstPage;
467                         lastPage := gTPrintHdl^^.prJob.iLstPage;
468
469                         gTPrintHdl^^.prJob.iFstPage := 1;
470                         gTPrintHdl^^.prJob.iLstPage := iPrPgMax;
471
472                         if (numberOfPages < lastPage) then
473                             lastPage := numberOfPages;
474
475                         for copy := 1 to numberOfCopies do
476                             begin
477                                 for page := firstPage to lastPage do
478                                     begin
479                                         if ((page - firstPage) mod iPFMaxPgs = 0) then
480                                             begin
481                                                 if (page <> firstPage) then
482                                                     begin
483                                                         PrCloseDoc(printPortPtr);
484
485                                                         if ((gTPrintHdl^^.prJob.bJDocLoop = bSpoolLoop) and (PrError = noErr)) then
486                                                             PrPicFile(gTPrintHdl, nil, nil, nil, printStatus);
487                                                         end;
488                                                         printPortPtr := PrOpenDoc(gTPrintHdl, nil, nil);
489                                                         end;
490                                                     if (PrError = noErr) then
491                                                         begin
492                                                             PrOpenPage(printPortPtr, nil);

```

```

493         if (PrError = noErr) then
494             DoDrawPage(gTPrintHdl ^^ . prInfo.rPage, page, numberOfPages);
495             PrClosePage(printPortPtr);
496         end;
497     end;
498
499     PrCloseDoc(printPortPtr);
500
501     if ((gTPrintHdl ^^ . prJob.bJDocLoop = bSpoolLoop) and (PrError = noErr)) then
502         PrPicFile(gTPrintHdl, nil, nil, nil, printStatus);
503     end;
504 end;
505 end;
506 end;
507
508 printError := PrError;
509
510 PrClose;
511
512 if ((printError <> noErr) and (printError <> iPrAbort)) then
513     DoPrintError(printError, false);
514
515 SetPort(oldPort);
516 DoActivateWindow;
517
518 end;
519 {of procedure PrintLoop}
520
521 { ##### DoMenuChoice }
522
523 procedure DoMenuChoice(menuChoice : longint);
524
525     var
526         menuID, menuItem : integer;
527         itemName : string;
528         daDriverRefNum : integer;
529
530     begin
531         menuID := HiWord(menuChoice);
532         menuItem := LoWord(menuChoice);
533
534         if (menuID = 0) then
535             Exit(DoMenuChoice);
536
537         case (menuID) of
538
539             mApple:
540                 begin
541                     GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
542                     daDriverRefNum := OpenDeskAcc(itemName);
543                     end;
544
545             mFile:
546                 begin
547                     if (menuItem = iPageSetup) then
548                         begin
549                             gInhibitPrintRecordsInfo := false;
550                             DoPrStyleDialog;
551                             end
552                     else if (menuItem = iPrint) then
553                         PrintLoop
554                     else if (menuItem = iQuit) then
555                         gDone := true;
556                     end;
557                 end;
558             {of case statement}
559
560             HiliteMenu(0);
561         end;
562     {of procedure DoMenuChoice}
563
564 { ##### DoMouseDown }
565
566 procedure DoMouseDown(eventRec : EventRecord);
567
568     var
569         myWindowPtr : WindowPtr;

```

```

570     partCode : integer;
571
572     begin
573     partCode := FindWindow(eventRec.where, myWindowPtr);
574
575     case (partCode) of
576
577         inMenuBar:
578             begin
579                 DoMenuChoice(MenuSelect(eventRec.where));
580             end;
581
582         inSysWindow:
583             begin
584                 SystemClick(eventRec, myWindowPtr);
585             end;
586
587         inContent:
588             begin
589                 if (myWindowPtr <> FrontWindow) then
590                     SelectWindow(myWindowPtr);
591                 end;
592
593         inDrag:
594             begin
595                 DragWindow(myWindowPtr, eventRec.where, qd.screenBits.bounds);
596             end;
597
598         inGoAway:
599             begin
600                 if (TrackGoAway(myWindowPtr, eventRec.where)) then
601                     gDone := true;
602                 end;
603             end;
604         {of case statement}
605     end;
606     {of procedure DoMouseDown}
607
608 { ##### DoEvents ##### }
609
610 procedure DoEvents(eventRec : EventRecord);
611
612     var
613     myWindowPtr : WindowPtr;
614     charCode : char;
615
616     begin
617     myWindowPtr := WindowPtr(eventRec.message);
618
619     case (eventRec.what) of
620
621         mouseDown:
622             begin
623                 DoMouseDown(eventRec);
624             end;
625
626         keyDown, autoKey:
627             begin
628                 charCode := chr(BAnd(eventRec.message, charCodeMask));
629                 if (BAnd(eventRec.modifiers, cmdKey) <> 0) then
630                     DoMenuChoice(MenuKey(charCode));
631                 end;
632
633         updateEvt:
634             begin
635                 BeginUpdate(myWindowPtr);
636                 EndUpdate(myWindowPtr);
637             end;
638
639         activateEvt:
640             begin
641                 DoActivateWindow;
642             end;
643         {of case statement}
644     end;
645     {of procedure DoEvents}
646

```

```

647 { ##### start of main program }
648
649
650 begin
651     { ..... initialize managers }
652
653     DoInitManagers;
654     gPrintRecordInitd := false;
655     gInhibitPrintRecordsInfo := false;
656
657     { ..... set up menu bar and menus }
658
659     menubarHdl := GetNewMBar(rMenubar);
660     if (menubarHdl = nil) then
661         ExitToShell;
662     SetMenuBar(menubarHdl);
663     DrawMenuBar;
664     menuHdl := GetMenuHandle(mApple);
665     if (menuHdl = nil)
666         then ExitToShell
667         else AppendResMenu(menuHdl, 'DRVVR');
668
669     { ..... open window }
670
671     gWindowPtr := GetNewWindow(rWindow, nil, WindowPtr(-1));
672     if (gWindowPtr = nil) then
673         ExitToShell;
674
675     SetPort(gWindowPtr);
676     TextSize(10);
677
678     { ..... load 'TEXT' and 'PICT' resources }
679
680     gTextHdl := GetResource('TEXT', rText);
681     if (gTextHdl = nil) then
682         ExitToShell;
683
684     gPictureHdl := GetPicture(rPicture);
685     if (gPictureHdl = nil) then
686         ExitToShell;
687
688     { ..... event loop }
689
690     gDone := false;
691
692     while not (gDone) do
693     begin
694         gotEvent := WaitNextEvent(everyEvent, eventRec, kMaxLong, nil);
695         if (gotEvent) then
696             DoEvents(eventRec);
697         end;
698     end.
699
700 { ##### }
701

```

## Demonstration Program Comments

---

When the program is run, the user should:

- Choose Page Setup... from the File menu, make changes in the style dialog, and observe the resulting contents of the main fields of the Tprint, TPrJob, TPrStyl, and TPrInfo records in the window.
- Choose Print... from the File menu, make changes in the job dialog, observe the results in the window, and observe the printout of the simulated document.

The user should print the simulated document several times using different page size, scaling, and orientation settings in the style dialog, and occasionally limiting the printout to one page only by changing the page range settings in the job dialog.

## **The constant declaration block**

Lines 46-55 establish constants related to menu IDs, menu item numbers and resources. The constant at Line 56 will be used to set the margins for the printout of a text document. Line 57 defines kMaxLong as the maximum possible longint value.

## **The variable declaration block**

gTPrintHdl will be assigned a handle to a TPrint record. gWindowPtr will be assigned the pointer to the window. gDone controls the exit from the main loop and thus program termination. gPrintRecordInitd will be set to true when a TPrint record has been created and initialised.

gInhibitPrintRecordsInfo is a flag which will prevent the display of information in the window in certain circumstances. gEditRecHdl will be assigned a handle to a TextEdit edit record. gTextHdl will be assigned a handle to the text used for printout. gPictureHdl will be assigned a handle to the picture used for printout.

## **The procedure DoPrintError**

DoPrintError is called from PrintLoop and DoPrStyleDialog if an error code is generated following a call to a Printing Manager routine. Depending on the nature of the error, either a Stop alert (Line 109) or a Caution alert (Line 113) is displayed, each containing the reported error code. In the case of a Stop alert, the program terminates when the user clicks the OK button (Line 110).

## **The function DoIsPrGeneralThere**

DoIsPrGeneralThere is called by DoGetPageOrientation to determine whether the current printer driver supports PrGeneral. The procedure is similar to that in DoGetPageOrientation, except that here we are interested only in the error code generated by a call to PrGeneral. If that error is the error represented by the constant resNotFound (-192), PrGeneral is not supported and false is returned (Line 136), otherwise true is returned (Line 138).

Note that, at Line 133, PrSetError is used to set the value in the low-memory global PrintErr to noErr in case PrGeneral generated an error code other than noErr. PrintErr holds the most recent Printing Manager error code and, since an actual printing error did not occur, it is necessary to ensure that PrintErr reflects that fact.

## **The function DoGetPageOrientation**

DoGetPageOrientation uses PrGeneral to establish the page orientation setting to be used for printing. A TGetRotnBlk record (Line 147) is used when PrGeneral is used to determine whether landscape orientation has been specified.

After establishing that the current printer driver supports PrGeneral (Line 150), the iopCode field of the TGetRotnBlk record is assigned the opcode getRtnOp and the hPrint field is assigned the handle to the TPrintRecord (Lines 152-153). PrGeneral is then called (Line 154) with the address of the TGetRotnBlk record as its argument.

Following the call, the fLandscape field of the TGetRotnBlk structure will contain true if landscape orientation has been selected. In this case (and assuming no errors), a value representing landscape orientation is returned to the calling function (Line 156), otherwise a value representing portrait orientation is returned (Line 157).

If the current printer driver does not support PrGeneral, a value representing this situation is returned (Line 160).

## **The procedure DoDrawPageOrientation**

DoDrawPageOrientation ascertains the page orientation selected by the user in the style dialog box and prints it in the window. It gets a value representing the orientation via a call to the application-defined function DoGetPageOrientation at Line 169.

## **The procedure DoPrintRecordsInfo**

DoPrintRecordsInfo extracts information from the TPrint, TPrInfo, TPrStyl and TPrJob records and prints it in the window. DoDrawRectStrings supports DoPrintRecords.

## **doActivateWindow**

doActivateWindow is called when an activate event is received. Its purpose is simply to redraw the text in the program's window when the style and job dialog boxes, and any other dialog or alert boxes presented by the system during printing operations, are dismissed. The flag gInhibitPrintRecordsInfo will defeat the drawing of this text if set to true.

## **The procedure DoDrawPage**

doDrawPage is called by printLoop to draw a specified page in the printing graphics port.

Lines 306-307 establish a rectangle equal to the received page rectangle less 180 pixels in width and 270 pixels in height. This smaller rectangle is centered on the page rectangle both laterally and vertically.

Line 309-310 calculate the number of lines of text that will fit into the height of that rectangle. Line 311 gets the total number of lines in the monostyled edit record created in the function doCalcNumberOfPages.

Lines 313-314 set the printing graphics port's font to Geneva 10 point (the same font and size used to calculate the number of pages).

Line 317 creates a new monostyled edit record with the rectangle established at Lines 306-307 passed in both the destination rectangle parameter and the view rectangle parameter. Line 318 gets a handle to the text in the monostyled edit record created in the function doCalcNumberOfPages.

Line 320 gets the starting offset, that is, the offset from the first character in the block of text to the first character in the first line of text for the specified page number. Lines 321-324 get the ending offset, that is, the offset to the last character in the last line of text for the specified page. Using these offsets, Line 327 then inserts the text for the page into the newly created edit record, an action which causes that text to be drawn in the printing graphics port.

If this is the first page, Lines 330-336 draw the previously loaded picture at the top left of the rectangle established at Lines 306-307.

Lines 338-340 draw the page number at the bottom left of that rectangle.

## **The procedure DoCalcNumberOfPages**

doCalcNumberOfPages is called by printLoop to calculate the number of pages in the (simulated) document.

The simulated document is provided by a 'TEXT' resource, which will be inserted into a TextEdit monostyled edit record. TextEdit is not addressed until Chapter 17 – Text and TextEdit; however, to facilitate an understanding of what is to follow, it is sufficient at this stage to understand that a monostyled edit record contains the following fields:

destRect	The destination rectangle into which text is drawn. The bottom of the destination rectangle can extend to accommodate the end of the text. In other words, you can think of the destination rectangle as bottomless.
viewRect	The rectangle within which text is actually displayed.
hText	A handle to the text.
lineHeight	The vertical spacing, in pixels, of the lines of text.
nLines	The total number of lines of text.
linestarts	An array with a number of elements corresponding to the number of lines of text. Each element contains the offset of the first character in each line.

Line 353 erases the window preparatory to the simulated document being drawn in the window.

Lines 355-366 establish a rectangle equal to the received page rectangle less 180 pixels in width and 270 pixels in height. (Note that this is the same size as the rectangle used in the drawing of each page in the printing graphics port.) Line 357 simply offsets this rectangle so that, when the document is drawn in the window, the top and right margins will be reduced to five pixels.

Lines 359-361 ensure that the window's font is set to Geneva 10 point.

Line 363 creates a new monostyled edit record with the rectangle established at Lines 355-357 passed in both the destination rectangle parameter and the view rectangle parameter. Line 364 inserts the previously loaded 'TEXT' resource into the edit record. The hText field of the edit record is now a handle to that text. The call to TEInsert also causes the text to be drawn in the window. (A 'TEXT' resource, rather than a 'TEXT' file, is used in this demonstration simply to keep that part of the source code that is not related to printing per se to a minimum.)

The matter of the actual calculation of the number of pages now follows. Line 366 gets the height of the rectangle established at Lines 355-356. Line 367 calculates how many lines of text will fit into that height. Line 368 then calculates the total number of rectangles (and thus the number of pages) required to accommodate the whole of the text.

Before the calculated number of pages is returned to the calling function (Line 375), Lines 370-373 draw the previously loaded picture at the top of the destination rectangle. This latter is simply to display the full contents of the top of the simulated document in the window. (Space for the picture is accounted for by the fact that the first 11 lines in the 'TEXT' resource are carriage returns.)

The edit record is retained because it will be used in the following function.

## **The procedure DoCreatePrintRecord**

DoCreatePrintRecord creates and initialises a TPrint record.

Memory is allocated at Line 387.

If the call to allocate memory is successful, Line 390 initialises the TPrint record to the system standard settings. If this call is successful, the global variable which indicates that an initialised TPrint record exists is set to true (Lines 392-393). The result of the PrError call is returned to the calling function (Line 394).

If the call to allocate memory is not successful, Line 397 simply closes down the program.

## **The procedure DoPrStyleDialog**

doPrStyleDialog is called when the user chooses Page Setup... from the File menu.

The call to PrOpen at Line 410 opens the Printing Manager and printer driver.

If the call is successful, Line 415 checks a new TPrint record currently exists. If not, doCreatePrintRecord is called to create a new Tprint record (Line 417). If doCreatePrintRecord does not return noErr (Line 418), Line 419 invokes a Stop alert. Line 421 opens the style dialog box.

If the call to PrOpen at Line 410 was not successful, a Caution alert is invoked (Lines 423-424).

Either way, Line 426 closes the Printing Manager (but not the printer driver), releasing the associated memory.

## **The procedure PrintLoop**

PrintLoop is the printing loop. It supports printers using deferred printing. However, it does not use a saved TPrint record (but rather creates one for the print job), and does not use a custom status dialog box and associated idle procedure. Also, it does not unload unneeded code segments at the beginning.

Line 444 saves a pointer to the current graphics port. Line 446 opens the Printing Manager, together with the current printer driver.

If the TPrint record has not already been created (Line 449), Line 450 calls an application-defined function to create and initialise a TPrint record. If this call is successful (Line 454), another application-defined function is called to calculate the number of pages (Line 456).

The job dialog box is then displayed (Line 458). If false is returned by the call to PrJobDialog (that is, the user clicked the Cancel button), the printing loop is bypassed. Otherwise, the first action is to retrieve the number of copies, the first page and the last page from the relevant fields of the TPrJob record (Lines 465-467). (Lines 461-463 are for demonstration program purposes only. Line 461 redraws the information in the window after the job dialog box disappears and Line 462 prints the selected page orientation in the bottom of the window.)

Since the only information which should be preserved between separate printings of the same document is that obtained via the style dialog box, the fields of the TPrJob record which store the first and last page numbers are set back to 1 and iPrPgMax (9999) respectively (Lines 469-470) before proceeding further.

If the last page number specified by the user exceeds the total number of pages in the document, the variable holding the last page value is set to the actual number of pages (Lines 472-473).

The copies loop is entered at Line 475 and the nested pages loop is entered at Line 477. The maximum number of pages that can be printed at a time is represented by the constant iPFMaxPgs (128). Lines 479 and 481 determine if this is the first or the 129th time around the pages loop. If it is the 129th (that is, 128 pages have been printed), Line 483 closes the printing graphics port and, if the printer driver is using deferred printing (Line 485), Line 486 sends the spool file to the printer. If this is either the first page of all or the first page after the first 128 have been printed, Line 488 initialises a new printing graphics port and makes it the current port.

For each page, Line 492 re-initialises the printing graphics port, the application-defined procedure DoDrawPage is called to draw that page's contents in the printing graphics port (Line 494), and Line 495 wraps up the printing of the current page.

When all pages have been printed, Line 499 closes the printing graphics port and, if the printer driver is using deferred printing (Line 501), Line 502 sends the spool file to the printer.

When all copies have been printed (or if control fell through to Line 508 as a result of an error), Line 510 releases memory associated with the Printing Manager (except the printer driver), and the result of a call to PrError at Line 508 is examined at Line 512. If an error occurred, and provided that error was not the error that is reported when the user (or the application) requests an abort, a Note alert is displayed advising the user of the error and error number (Line 513).

Finally, the saved graphics port is restored (Line 515) and the window is activated (Line 516).

## **The procedure DoMenuChoice**

DoMenuChoice handles menu choices from the Apple and File menus.

Note that, if the user chooses Page Setup... from the File menu, the application-defined function DoPrStyleDialog is called. Note also that, if the user chooses Print... from the File menu, the application-defined function printLoop is called.

## **The procedures DoMouseDown and DoEvents**

DoEvents and DoMouseDown perform minimal event handling consistent with the satisfactory performance of the demonstration aspects of the program. Note that, at Line 598, an activate event results in a call to the procedure DoActivateWindow.

## **The main program block**

The main function initialises the system software managers (Line 654), sets up the menus (Lines 660-668), opens a window (Line 672), sets the window's graphics port as the current port (Line 676), sets the text size to 10 (Line 677), loads a 'TEXT' resource and a 'PICT' resource (Lines 681-687), and enters the main event loop (Lines 691-698).

Note that, in this program, error handling of all errors other than Printing Manager errors is somewhat rudimentary. The program simply exits.