13 Version 1.1

PRINTING

Includes Demonstration Program Printing

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**[1] 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.

[1]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.

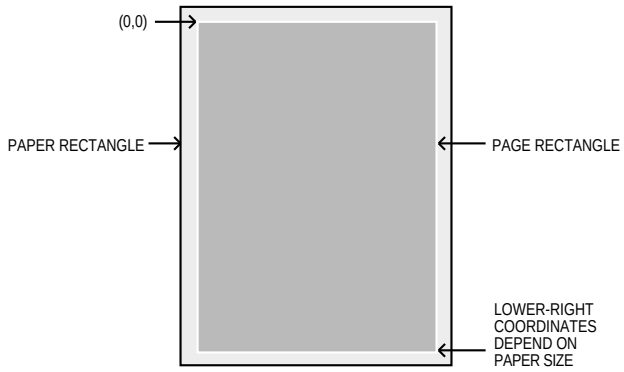

FIG 1 - PAPER AND PAGE RECTANGLES

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.

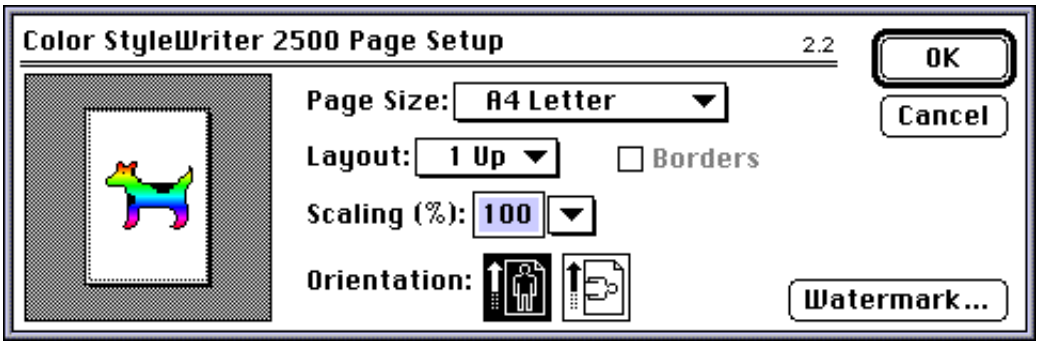

FIG 2 - STYLE DIALOG BOX FOR STYLEWRITER II PRINTER

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.

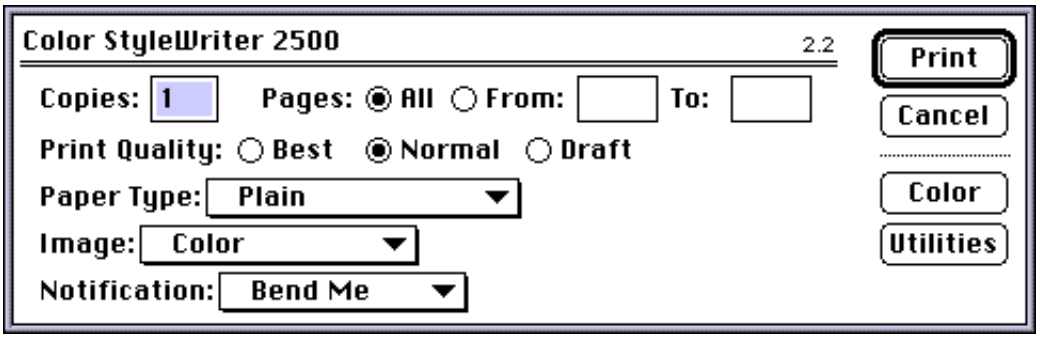

FIG 3 - JOB DIALOG BOX FOR STYLEWRITER II PRINTER

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.

You 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.

```
struct TPrint
{
short    iPrVersion;
TPrInfo  prInfo; // Printer information record.
Rect     rPaper; // Paper rectangle.
TPrStl   prStl;   // Printing style record.
TPrInfo  prInfoPT;
TPrXInfo prXInfo;
TPrJob   prJob;  // Printing job record.
 short printX[19];
};
```

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.

```
struct TPrJob
{
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;
PrIdleProcPtrpIdleProc;// Pointer to idle procedure.
StringPtr   pFileName; // Spool filename.
short       iFileVol;  // Spool file volume.
char        bFileVers; // Spool file version.
char        bJobX;
};
typedef struct TPrJob TPrJob;
typedef TPrJob *TPPrJob;
```

```
struct TPrStl
{
short wDev;    // Device number of printer.
short iPageV;
short iPageH;
char  bPort;
TFeed feed;    // Feed type.
};
typedef struct TPrStl TPrStl;
typedef TPrStl *TPPrStl;
```

```
struct TPrInfo
{
short iDev;
short iVRes;  // Vertical resolution in dpi.
short iHRes;  // Horizontal resolution in dpi.
Rect  rPage;  // Page rectangle.
};
typedef struct TPrInfo TPrInfo;
typedef TPrInfo *TPPrInfo;
```

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:

```
struct TPrPort
{
  GrafPort  gPort;   // Printer's graphics port record.
  QDProcs   gProcs;  // Procedures for printing in the graphics port.
  ...                // More fields for internal use.
};

typedef struct TPrPort TPrPort;
typedef TPrPort *TPPrPort;
```

Field Descriptions

`gPort` A graphics port record, which may be either a `CGrafPort` or a `GrafPort` record, depending on whether the current printer supports colour and greyscale, and whether Color QuickDraw is available on the computer.[2]

`gProcs` A `QDProcs` record, which contains pointers to routines which the printer driver may have designated to take the place of QuickDraw routines.

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.

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[3] 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).

---

[2] 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.

[3] 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[4] 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).[5]

    - **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.)

---

[4] The `PrDlgMain` function is used to display a customized job dialog box.

[5] The value represented by `iPFMaxPgs` is `128`.

```
THPrint tPrintHdl;
...

tPrintHdl = (THPrint) NewHandleClear(sizeof(TPrint));
if(tPrintHdl != NULL )
{
  PrintDefault(tPrintHdl);    // Sets appropriate default values for current driver.
  if(printError = PrError())
    doPrintError(printError);
}
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.

```
OSErr  doGetPrintRecord(SInt16 refNum,THPrint tPrintHdl,Boolean *prRecChanged)
{
  SInt16 saveResFile;
  OSErr result;

  saveResFile = CurResFile();
  UseResFile(refNum};

  tPrintHdl = (THPrint) Get1Resource('SPRC',kDocPrintRec);
  if(tPrintHdl != NULL)
  {
    DetachResource((Handle) tPrintHdl);
    prRecChanged = PrValidate(tPrintHdl); // Check validity of TPrint record.
    UseResFile(saveResFile);
    return(PrError());
  }
  else
  {
    UseResFile(saveResFile);
    return(kNilHandlePrintErr);
  }
}
```

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:

```
void  printLoop(DocumentRecordHld docToPrint,Boolean displayJobDialog)
{
  GrafPtr   oldPort;
  SInt16    numberOfPages, numberOfCopies;
  Boolean   userClickedOK;
  SInt16    firstPage, lastPage, copy, page;
  TPrStatus tprStatus;
  SInt16    printError;

  GetPort(&oldPort);
  doUnloadSegments();

  PrOpen();
  if(PrError() == noErr)
  {
    gPrintResFile = CurResFile();
    gTPrintHdl = (*docToPrint)->docPrintRecordHdl;.
    changed = PrValidate(gTPrintHdl);

    if(PrError() == noErr)
    {
      numberOfPages = doCalulateNumberOfPages((*gTPrintHdl)->prInfo.rPage);

      if(displayJobDialog)
        userClickedOK = PrJobDialog(gTPrintHdl);
      else
        userClickedOK = doJobMerge(gTPrintHdl);

      if(userClickedOK)
      {
        numberOfCopies = (*gTPrintHdl)->prJob.iCopies;

        firstPage = (*gTPrintHdl)->prJob.iFstPage;
        lastPage = (*gTPrintHdl)->prJob.iLstPage;

        (*gTPrintHdl)->prJob.iFstPage = 1;
        (*gTPrintHdl)->prJob.iLstPage = iPrPgMax;
```

```
        if(numberOfPages < lastPage)
          lastPage = numberOfPages;

        doActivateFrontWindow(false,oldPort);                              // Optional
        gPrintStatusDlg = GetNewDialog(rPrintStatus,NULL,(WindowPtr) -1);  // Optional
        doDialogBoxItems(docToPrint);                                      // Optional
        ShowWindow(gPrintStatusDlg);                                       // Optional
        (*gTPrintHdl)->prJob.pIdleProc = &doPrintIdle;                     // Optional

        for(copy=1;copy<numberOfCopies+1;copy++)
        {
          UseResFile(gPrintResFile);

          for(page=firstPage;page<lastPage+1;page++)
          {
            if((page - firstPage) % iPFMaxPgs == 0)
            {
              if(page != firstPage)
              {
                PrCloseDoc(gPrintPortPtr);

                if(((*gTPrintHdl)->prJob.bJDocLoop == bSpoolLoop) && (PrError() == noErr))
                  PrPicFile(gTPrintHdl,NULL,NULL,NULL,&tprStatus);
              }
              gPrintPortPtr = PrOpenDoc(gTPrintHdl,NULL,NULL);
            }
            if(PrError() == noErr)
            {
              PrOpenPage(gPrintPortPtr,NULL);
              if(PrError() == noErr)
              {
                doDrawPrintPage((*gTPrintHdl)->prInfo.rPage,docToPrint,
                                (GrafPtr) gPrintPortPtr,page);
              }
              PrClosePage(gPrintPortPtr);
            }
          }

          PrCloseDoc(gPrintPortPtr);

          if (((*gTPrintHdl)->prJob.bJDocLoop == bSpoolLoop) && (PrError() == noErr))
            PrPicFile(gTPrintHdl,NULL,NULL,NULL,&tprStatus);
        }
      }
    }
  }

  printError = PrError();

  PrClose();

  if(printError != noErr)
    doPrintError(printError);

  DisposeDialog(gPrintStatusDlg);
  SetPort(oldPort);
  doActivateFrontWindow(true,oldPort);
}
```

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 "print status" modal dialog box to display printing status information:

```
pascal void  doPrintIdle(void)
{
  GrafPtr      oldPort;
  EventRecord  eventRec;
  Boolean      gotEvent;
  SInt16       itemHit;
  Boolean      handled, cancelled;

  GetPort(&oldPort);
  SetPort(gPrintStatusDlg);

  gotEvent == WaitNextEvent(everyEvent,&eventRec,15,NULL);

  if(gotEvent)
  {
    // 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,&eventRec,&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();
        if(cancelled)
          itemHit = kStopButton;

        // To handle hits in the "print status" dialog, doHandleHitsInStatusBox should
        // 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 wristwatch cursor.

        handled = doHandleHitsInStatusBox(itemHit);
      };

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

      doUpdateStatusInformation(cancelled);

      SetPort(oldPort);
    }
```

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[6], 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 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

[6]See Chapter 15 — More on Resources.

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`.

```
Boolean  doIsPrGeneralThere(void)
{
  TGetRotnBlk getRotRec;
  OSErr       printError;

  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 = PrError();
  PrSetError(noErr);

  if(printError == resNotFound)
    return(false);
  else
    return(true);
}
```

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:

```
SInt16 doGetPageOrientation(void)
{
  TGetRotnBlk getRotRec;

  if(doIsPrGeneralThere)
  {
    getRotRec.iOpCode = getRotnOp;
    getRotRec.hPrint = gTPrintHdl;
    PrGeneral((Ptr) &getRotRec);
    if((getRotRec.iError == noErr) && (PrError() == noErr) && getRotRec.fLandscape)
      return(kInLandscapeOrientation);
    else
      return(kInPortraitOrientation);
  }
  else
    return(kPrGeneralAbsent);
}
```

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[7] 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 noticeaby different.

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.

---

[7] A glyph is the visual representation of a character. See Chapter 17 —– Text. and TextEdit.

- 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

```
iPFMaxPgs       = 128   Maximum pages in spool file.
iPrPgFract      = 120   Page scale factor.
iPrPgFst        = 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

```
struct TPrint
{
  short           iPrVersion;   // (Reserved)
  TPrInfo         prInfo;       // PrInfo data associated with the current style.
  Rect            rPaper;       // Paper rectangle (offset from rPage).
  TPrStl          prStl;        // This print request's style.
  TPrInfo         prInfoPT;     // (Reserved)
  TPrXInfo        prXInfo;      // (Reserved)
  TPrJob          prJob;        // Print Job request.
 short            printX[19];   // (Reserved)
};

typedef struct TPrint TPrint;
typedef TPrint *TPPrint, **THPrint;
```

Printer Information Record

```
struct TPrInfo
{
  short           iDev;         // (Reserved)
  short           iVRes;        // Vertical resolution of printer in dpi.
  short           iHRes;        // Horizontal resolution of printer in dpi.
  Rect            rPage;        // Page (printable) rectangle in device coordinates.
};

typedef struct TPrInfo TPrInfo;
typefdef TPrInfo *TPPrInfo;
```

Print Job Record

```
struct TPrJob
{
  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 file name: NULL 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;
```

Printing Style Record

```
struct TPrStl
{
  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;
```

Printing Graphics Port Record

```
struct TPrPort
{
  GrafPort        gPort;        // Graphics port for printing.
  QDProcs         gProcs;       // Procedures for printing in 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;
```

Printing Status Record

```
struct TPrStatus
{
  short           iTotPages;    // Total pages in print File.
  short           iCurPage;     // Current page number.
  short           iTotCopies;   // Current 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 printer record.
  TPPrPort        pPrPort;      // Pointer to the active printing graphics port.
  PicHandle       hPic;         // Handle to the active picture.
};

typedef struct TPrStatus TPrStatus;
typedef TPrStatus *TPPrStatus;
```

Print Dialog Box Record

```
struct TPrDlg
{
  DialogRecord    Dlg;          // A dialog record.
  ModalFilterUPP  pFltrProc;    // The filter proc.
  PItemUPP        pItemProc;    // The item evaluating proc.
  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);
```

Page Orientation Record

```
struct TGetRotnBlk
{
  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.
  SInt8           bXtra;        // (Reserved)
};

typedef Rect *TPRect;
typedef pascal void (*PrIdleProcPtr)(void);
typedef pascal void (*PItemProcPtr)(DialogPtr theDialog, short item);
```

Routines

Opening and Closing the Printing Manager

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

Initialising and Validating `TPrint` Records

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

Displaying and Customising Print Dialog Boxes

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

Printing a Document

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

Optimising Printing

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

Handling Printing Errors

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

Demonstration Program

```
1  // ##############################################################################################
2  // Printing.c
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 out 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 // …… includes
33
34 #include <Fonts.h>
35 #include <Menus.h>
36 #include <TextEdit.h>
37 #include <Dialogs.h>
38 #include <SegLoad.h>
```

```
#include <ToolUtils.h>
#include <Devices.h>
#include <Resources.h>
#include <Printing.h>

// ……… defines

#define mApple      128
#define mFile       129
#define  iQuit      11
#define  iPageSetup 8
#define  iPrint     9
#define rMenubar    128
#define rWindow     128
#define rText       128
#define rPicture    128
#define rPrintAlert 128
#define kMargin     90
#define MAXLONG     0x7FFFFFFF

// ……… global variables

THPrint   gTPrintHdl;
WindowPtr gWindowPtr;
Boolean   gDone;
Boolean   gPrintRecordInited = false;
Boolean   gInhibitPrintRecordsInfo = false;
TEHandle  gEditRecHdl;
Handle    gTextHdl;
PicHandle gPictureHdl;

// ……… function prototypes

void    main                (void);
void    doInitManagers      (void);
void    doEvents            (EventRecord *);
void    doMouseDown         (EventRecord *);
void    doActivateWindow    (void);
void    doMenuChoice        (long);
void    printLoop           (void);
OSErr   doCreatePrintRecord (void);
void    doPrStyleDialog     (void);
SInt16  doCalcNumberOfPages (Rect);
void    doDrawPage          (Rect,SInt16,SInt16);
void    doPrintRecordsInfo  (void);
void    doDrawRectStrings   (Str255,SInt16,SInt16,Str255,SInt16,SInt16,Str255);
void    doDrawPageOrientation (void);
SInt16  doGetPageOrientation  (void);
Boolean doIsPrGeneralThere  (void);
void    doPrintError        (SInt16,Boolean);

// ############################################################################## main

void  main(void)
{
  Handle      menubarHdl;
  MenuHandle  menuHdl;
  EventRecord eventRec;
  Boolean     gotEvent;

  // ……… initialise managers

  doInitManagers();

  // ……… set up menu bar and menus

  if(!(menubarHdl = GetNewMBar(rMenubar)))
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  if(!(menuHdl = GetMenuHandle(mApple)))
    ExitToShell();
  else
    AppendResMenu(menuHdl,'DRVR');

  // ……… open window
```

```
116
117   if(!(gWindowPtr = GetNewWindow(rWindow,NULL,(WindowPtr)-1)))
118     ExitToShell();
119
120   SetPort(gWindowPtr);
121   TextSize(10);
122
123   // ……… load 'TEXT' and 'PICT' resources
124
125   gTextHdl = GetResource('TEXT',rText);
126   if(gTextHdl == NULL)
127     ExitToShell();
128
129   gPictureHdl = GetPicture(rPicture);
130   if(gPictureHdl == NULL)
131     ExitToShell();
132
133   // ……… event loop
134
135   gDone = false;
136
137   while(!gDone)
138   {
139     gotEvent = WaitNextEvent(everyEvent,&eventRec,MAXLONG,NULL);
140     if(gotEvent)
141       doEvents(&eventRec);
142   }
143 }
144
145 // ########################################################################## doInitManagers
146
147 void  doInitManagers(void)
148 {
149   MaxApplZone();
150   MoreMasters();
151
152   InitGraf(&qd.thePort);
153   InitFonts();
154   InitWindows();
155   InitMenus();
156   TEInit();
157   InitDialogs(NULL);
158
159   InitCursor();
160   FlushEvents(everyEvent,0);
161 }
162
163 // ############################################################################## doEvents
164
165 void  doEvents(EventRecord *eventRecPtr)
166 {
167   WindowPtr windowPtr;
168   UInt8     charCode;
169
170   windowPtr = (WindowPtr) eventRecPtr->message;
171
172   switch(eventRecPtr->what)
173   {
174     case mouseDown:
175       doMouseDown(eventRecPtr);
176       break;
177
178     case keyDown:
179     case autoKey:
180       charCode = eventRecPtr->message & charCodeMask;
181       if((eventRecPtr->modifiers & cmdKey) != 0)
182         doMenuChoice(MenuKey(charCode));
183       break;
184
185     case updateEvt:
186       BeginUpdate(windowPtr);
187       EndUpdate(windowPtr);
188       break;
189
190     case activateEvt:
191       doActivateWindow();
192       break;
```

```
193     }
194   }
195
196   // ############################################################################## doMouseDown
197
198   void  doMouseDown(EventRecord *eventRecPtr)
199   {
200     WindowPtr windowPtr;
201     SInt16    partCode;
202
203     partCode = FindWindow(eventRecPtr->where,&windowPtr);
204
205     switch(partCode)
206     {
207       case inMenuBar:
208         doMenuChoice(MenuSelect(eventRecPtr->where));
209         break;
210
211       case inSysWindow:
212         SystemClick(eventRecPtr,windowPtr);
213         break;
214
215       case inContent:
216         if(windowPtr != FrontWindow())
217           SelectWindow(windowPtr);
218         break;
219
220       case inDrag:
221         DragWindow(windowPtr,eventRecPtr->where,&qd.screenBits.bounds);
222         break;
223
224       case inGoAway:
225         if(TrackGoAway(windowPtr,eventRecPtr->where) == true)
226           gDone = true;
227         break;
228     }
229   }
230
231   // ############################################################################ doActivateWindow
232
233   void  doActivateWindow(void)
234   {
235     if(FrontWindow() == gWindowPtr && gPrintRecordInited && !gInhibitPrintRecordsInfo)
236       doPrintRecordsInfo();
237   }
238
239   // ################################################################################ doMenuChoice
240
241   void  doMenuChoice(long menuChoice)
242   {
243     SInt16  menuID, menuItem;
244     Str255  itemName;
245     SInt16  daDriverRefNum;
246
247     menuID = HiWord(menuChoice);
248     menuItem = LoWord(menuChoice);
249
250     if(menuID == 0)
251       return;
252
253     switch(menuID)
254     {
255       case mApple:
256         GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
257         daDriverRefNum = OpenDeskAcc(itemName);
258         break;
259
260       case mFile:
261         if(menuItem == iPageSetup)
262         {
263           gInhibitPrintRecordsInfo = false;
264           doPrStyleDialog();
265         }
266         else if(menuItem == iPrint)
267           printLoop();
268         else if(menuItem == iQuit)
269           gDone = true;
```

```
        break;
    }

    HiliteMenu(0);
}

// ############################################################################## printLoop

void  printLoop(void)
{
    GrafPtr   oldPort;
    SInt16    printError;
    SInt16    numberOfPages, numberOfCopies;
    Boolean   userClickedOK;
    SInt16    firstPage, lastPage, copy, page;
    TPPrPort  printPortPtr;
    TPrStatus tprStatus;

    GetPort(&oldPort);

    PrOpen();
    if(PrError() == noErr)
    {
      if(!gPrintRecordInited)
        printError = doCreatePrintRecord();
      else
        printError = noErr;

      if(printError == noErr)
      {
        numberOfPages = doCalcNumberOfPages((*gTPrintHdl)->prInfo.rPage);

        userClickedOK = PrJobDialog(gTPrintHdl);
        if(userClickedOK)
        {
          doPrintRecordsInfo();
          doDrawPageOrientation();
          gInhibitPrintRecordsInfo = true;

          numberOfCopies = (*gTPrintHdl)->prJob.iCopies;
          firstPage = (*gTPrintHdl)->prJob.iFstPage;
          lastPage = (*gTPrintHdl)->prJob.iLstPage;

          (*gTPrintHdl)->prJob.iFstPage = 1;
          (*gTPrintHdl)->prJob.iLstPage = iPrPgMax;

          if(numberOfPages < lastPage)
            lastPage = numberOfPages;

          for(copy=1;copy<numberOfCopies+1;copy++)
          {
            for(page=firstPage;page<lastPage+1;page++)
            {
              if((page - firstPage) % iPFMaxPgs == 0)
              {
                if(page != firstPage)
                {
                  PrCloseDoc(printPortPtr);

                  if(((*gTPrintHdl)->prJob.bJDocLoop == bSpoolLoop) && (PrError() == noErr))
                    PrPicFile(gTPrintHdl,NULL,NULL,NULL,&tprStatus);
                }
                printPortPtr = PrOpenDoc(gTPrintHdl,NULL,NULL);
              }
              if(PrError() == noErr)
              {
                PrOpenPage(printPortPtr,NULL);
                if(PrError() == noErr)
                  doDrawPage((*gTPrintHdl)->prInfo.rPage,page,numberOfPages);
                PrClosePage(printPortPtr);
              }
            }

            PrCloseDoc(printPortPtr);

            if(((*gTPrintHdl)->prJob.bJDocLoop == bSpoolLoop) && (PrError() == noErr))
              PrPicFile(gTPrintHdl,NULL,NULL,NULL,&tprStatus);
```

```
        }
      }
    }
  }

  printError = PrError();

  PrClose();

  if(printError != noErr && printError != iPrAbort)
    doPrintError(printError,false);

  SetPort(oldPort);
  doActivateWindow();
}

// ########################################################################## doCreatePrintRecord

OSErr  doCreatePrintRecord(void)
{
  SInt16  printError;

  gTPrintHdl = (THPrint) NewHandleClear(sizeof(TPrint));
  if(gTPrintHdl != NULL )
  {
    PrintDefault(gTPrintHdl);
    printError = PrError();
    if(printError == noErr)
      gPrintRecordInited = true;
    return(printError);
  }
  else
    ExitToShell();
}

// ########################################################################## doPrStyleDialog

void  doPrStyleDialog(void)
{
  SInt16  printError;

  PrOpen();

  printError = PrError();
  if(printError == noErr)
  {
    if(!gPrintRecordInited)
    {
      printError = doCreatePrintRecord();
      if(printError != noErr)
        doPrintError(printError,true);
    }

    PrStlDialog(gTPrintHdl);
  }
  else
    doPrintError(printError,false);

  PrClose();
}

// ########################################################################## doCalcNumberOfPages

SInt16  doCalcNumberOfPages(Rect pageRect)
{
  Rect    destRect, pictureRect;
  SInt16  fontNum, heightDestRect, linesPerPage, numberOfPages;

  EraseRect(&(gWindowPtr->portRect));

  SetRect(&destRect,pageRect.left + kMargin,pageRect.top + (kMargin * 1.5),
          pageRect.right - kMargin,pageRect.bottom - (kMargin * 1.5));
  OffsetRect(&destRect,- (kMargin - 5),- ((kMargin * 1.5) - 5));

  GetFNum("\pGeneva",&fontNum);
  TextFont(fontNum);
  TextSize(10);
```

```
424
425     gEditRecHdl = TENew(&destRect,&destRect);
426     TEInsert(*gTextHdl,GetHandleSize(gTextHdl),gEditRecHdl);
427
428     heightDestRect = destRect.bottom - destRect.top;
429     linesPerPage = heightDestRect / (*gEditRecHdl)->lineHeight;
430     numberOfPages = ((*gEditRecHdl)->nLines / linesPerPage) + 1;
431
432     SetRect(&pictureRect,destRect.left,destRect.top,
433           destRect.left + ((*gPictureHdl)->picFrame.right - (*gPictureHdl)->picFrame.left),
434           destRect.top + ((*gPictureHdl)->picFrame.bottom - (*gPictureHdl)->picFrame.top));
435     DrawPicture(gPictureHdl,&pictureRect);
436
437     return(numberOfPages);
438   }
439
440   // ############################################################################## doDrawPage
441
442   void  doDrawPage(Rect pageRect,SInt16 pageNumber,SInt16 numberOfpages)
443   {
444     Rect      destRect, pictureRect;
445     SInt16    heightDestRect, linesPerPage, numberOfLines, fontNum;
446     TEHandle  pageEditRecHdl;
447     Handle    textHdl;
448     SInt32    startOffset, endOffset;
449     Str255    theString;
450
451     SetRect(&destRect,pageRect.left + kMargin,pageRect.top + (kMargin * 1.5),
452             pageRect.right - kMargin,pageRect.bottom - (kMargin * 1.5));
453
454     heightDestRect = destRect.bottom - destRect.top;
455     linesPerPage = heightDestRect / (*gEditRecHdl)->lineHeight;
456     numberOfLines = (*gEditRecHdl)->nLines;
457
458     GetFNum("\pGeneva",&fontNum);
459     TextFont(fontNum);
460     TextSize(10);
461
462     pageEditRecHdl = TENew(&destRect,&destRect);
463     textHdl = (*gEditRecHdl)->hText;
464
465     startOffset = (*gEditRecHdl)->lineStarts[(pageNumber - 1) * linesPerPage];
466     if(pageNumber == numberOfpages)
467       endOffset = (*gEditRecHdl)->lineStarts[numberOfLines];
468     else
469       endOffset = (*gEditRecHdl)->lineStarts[pageNumber * linesPerPage];
470
471     HLock(textHdl);
472     TEInsert(*textHdl + startOffset,endOffset - startOffset,pageEditRecHdl);
473     HUnlock(textHdl);
474
475     if(pageNumber == 1)
476     {
477       SetRect(&pictureRect,destRect.left,destRect.top,
478           destRect.left + ((*gPictureHdl)->picFrame.right - (*gPictureHdl)->picFrame.left),
479           destRect.top + ((*gPictureHdl)->picFrame.bottom - (*gPictureHdl)->picFrame.top));
480
481       DrawPicture(gPictureHdl,&pictureRect);
482     }
483
484     MoveTo(destRect.left,pageRect.bottom - 25);
485     NumToString((SInt32) pageNumber,theString);
486     DrawString(theString);
487   }
488
489   // ########################################################################## doPrintRecordsInfo
490
491   void  doPrintRecordsInfo(void)
492   {
493     Str255  s2, s3;
494
495     EraseRect(&(gWindowPtr->portRect));
496
497     MoveTo(20,25);
498     DrawString("\pFrom TPrint, TPrInfo and TPrStl records:");
499
500     NumToString((long) (*gTPrintHdl)->rPaper.top,s2);
```

```
501     NumToString((long) (*gTPrintHdl)->rPaper.left,s3);
502     doDrawRectStrings("\pPaper Rectangle (top,left):",20,45,s2,190,45,s3);
503
504     NumToString((long) (*gTPrintHdl)->rPaper.bottom,s2);
505     NumToString((long) (*gTPrintHdl)->rPaper.right,s3);
506     doDrawRectStrings("\pPaper Rectangle (bottom,right):",20,60,s2,190,60,s3);
507
508     NumToString((long) ((*gTPrintHdl)->prInfo.rPage).top,s2);
509     NumToString((long) ((*gTPrintHdl)->prInfo.rPage).left,s3);
510     doDrawRectStrings("\pPage Rectangle (top,left):",20,75,s2,190,75,s3);
511
512     NumToString((long) ((*gTPrintHdl)->prInfo.rPage).bottom,s2);
513     NumToString((long) ((*gTPrintHdl)->prInfo.rPage).right,s3);
514     doDrawRectStrings("\pPage Rectangle (bottom,right):",20,90,s2,190,90,s3);
515
516     MoveTo(20,105);
517     DrawString("\pFeed Type:");
518     MoveTo(190,105);
519     if((*gTPrintHdl)->prStl.feed == 0)
520       DrawString("\pCut sheet");
521     else if((*gTPrintHdl)->prStl.feed == 1)
522       DrawString("\pFanfold");
523
524     MoveTo(20,120);
525     DrawString("\pVertical resolution:");
526     NumToString((long) (*gTPrintHdl)->prInfo.iVRes,s2);
527     MoveTo(190,120);
528     DrawString(s2);
529
530     MoveTo(20,135);
531     DrawString("\pHorizontal resolution:");
532     NumToString((long) (*gTPrintHdl)->prInfo.iHRes,s2);
533     MoveTo(190,135);
534     DrawString(s2);
535
536     MoveTo(20,155);
537     DrawString("\pFrom TPrJob Record:");
538
539     MoveTo(20,175);
540     DrawString("\pFirst Page:");
541     NumToString((long) (*gTPrintHdl)->prJob.iFstPage,s2);
542     MoveTo(190,175);
543     DrawString(s2);
544
545     MoveTo(20,190);
546     DrawString("\pLast Page:");
547     NumToString((long) (*gTPrintHdl)->prJob.iLstPage,s2);
548     MoveTo(190,190);
549     DrawString(s2);
550
551     MoveTo(20,205);
552     DrawString("\pNumber of Copies:");
553     NumToString((long) (*gTPrintHdl)->prJob.iCopies,s2);
554     MoveTo(190,205);
555     DrawString(s2);
556
557     MoveTo(20,225);
558     DrawString("\pNote:  Some printer drivers always set the iCopies field of the TPrJob");
559     MoveTo(20,240);
560     DrawString("\precord to 1 and handle multiple copies internally.");
561   }
562
563   // ########################################################################## doDrawRectStrings
564
565   void  doDrawRectStrings(Str255 s1,SInt16 x1,SInt16 y1,Str255 s2,SInt16 x2,SInt16 y2,Str255 s3)
566   {
567     MoveTo(x1,y1);
568     DrawString(s1);
569     MoveTo(x2,y2);
570     DrawString("\p(");
571     DrawString(s2);
572     DrawString("\p,");
573     DrawString(s3);
574     DrawString("\p)");
575   }
576
577   // ########################################################################## drawPageOrientation
```

```
578
579 void  doDrawPageOrientation(void)
580 {
581   SInt16  orientation;
582
583   MoveTo(20,260);
584   DrawString("\pOrientation selected:");
585
586   orientation = doGetPageOrientation();
587
588   MoveTo(190,260);
589   if(orientation == 1)
590     DrawString("\pLandscape");
591   else if(orientation == 2)
592     DrawString("\pPortrait");
593   else
594     DrawString("\p(PrGeneral not supported by driver)");
595 }
596
597 // ##################################################################### doGetPageOrientation
598
599 SInt16  doGetPageOrientation(void)
600 {
601   TGetRotnBlk getRotRec;
602
603   if(doIsPrGeneralThere)
604   {
605     getRotRec.iOpCode = getRotnOp;
606     getRotRec.hPrint = gTPrintHdl;
607     PrGeneral((Ptr) &getRotRec);
608     if((getRotRec.iError == noErr) && (PrError() == noErr) && getRotRec.fLandscape)
609       return(1);
610     else
611       return(2);
612   }
613   else
614     return(3);
615 }
616
617 // ##################################################################### doIsPrGeneralThere
618
619 Boolean  doIsPrGeneralThere(void)
620 {
621   TGetRotnBlk getRotRec;
622   OSErr       printError;
623
624   printError = 0;
625   getRotRec.iOpCode = getRotnOp;
626   getRotRec.hPrint = gTPrintHdl;
627
628   PrGeneral((Ptr) &getRotRec);
629
630   printError = PrError();
631   PrSetError(noErr);
632
633   if(printError == resNotFound)
634     return(false);
635   else
636     return(true);
637 }
638
639 // ############################################################################## doPrintError
640
641 void  doPrintError(SInt16 printError,Boolean fatal)
642 {
643   Str255 errorNumberString;
644
645   NumToString((long) printError,errorNumberString);
646   ParamText(errorNumberString,NULL,NULL,NULL);
647   if(fatal)
648   {
649     StopAlert(rPrintAlert,NULL);
650     ExitToShell();
651   }
652   else
653     CautionAlert(rPrintAlert,NULL);
654 }
```

```
655
656 // ############################################################################################
```

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.

#define

Lines 46-55 establish constants related to menu IDs, menu item numbers and resources. Line 57 defines MAXLONG as the maximum possible long value.

Global Variables

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. gPrintRecordInited 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.

main

The main function initialises the system software managers (Line 101), sets up the menus (Lines 105-113), opens a window (Line 117), sets the window's graphics port as the current port (Line 120), sets the text size to 10 (Line 121), loads a 'TEXT' resource and a 'PICT' resource (Lines 125-131), and enters the main event loop (Lines 135-142).

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

doEvents and doMouseDown

doEvents and doMouseDown perform minimal event handling consistent with the satisfactory performance of the demonstration aspects of the program. Note that, at Lines 190-192, an activate event results in a call to the function doActivateWindow.

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.

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 (Lines 266-270). Note also that, if the user chooses Print… from the File menu, the application-defined function printLoop is called (Lines 266-267).

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 288 saves a pointer to the current graphics port. Line 290 opens the Printing Manager, together with the current printer driver.

If the TPrint record has not already been created (Line 293), Line 294 calls an application-defined function to create and initialise a TPrint record. If this call is successful (Line 298), another application-defined function is called to calculate the number of pages (Line 300.

The job dialog box is then displayed (Line 302). 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 309-311). (Lines 305-307 are for demonstration program purposes only. Line 305 redraws the information in the window after the job dialog box disappears and Line 306 prints the selected page orientation in the bottom of the window.)

Since the only information that 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 313-314) 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 316-317).

The copies loop is entered at Line 319 and the nested pages loop is entered at Line 321. The maximum number of pages that can be printed at a time is represented by the constant iPFMaxPgs (128). Lines 323 and 325 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 327 closes the printing graphics port and, if the printer driver is using deferred printing (Line 329), Line 330 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 332 initialises a new printing graphics port and makes it the current port.

For each page, Line 336 re-initialises the printing graphics port, the application-defined function doDrawPage is called to draw that page's contents in the printing graphics port (Line 338), and Line 339 wraps up the printing of the current page.

When all pages have been printed, Line 343 closes the printing graphics port and, if the printer driver is using deferred printing (Line 345), Line 346 sends the spool file to the printer.

When all copies have been printed (or if control fell through to Line 352 as a result of an error), Line 354 releases memory associated with the Printing Manager (except the printer driver), and the result of a call to PrError at Line 352 is examined at Line 356. 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 357).

Finally, the saved graphics port is restored (Line 359) and the window is activated (Line 360).

doCreatePrintRecord

doCreatePrintRecord creates and initialises a TPrint record.

Memory is allocated at Line 369.

If the call to allocate memory is successful, Line 372 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 373-375). The result of the PrError call is returned to the calling function at Line 376.

If the call to allocate memory is not successful, Line 379 simply closes down the program.

doPrStyleDialog

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

The call to PrOpen at Line 388 opens the Printing Manager and printer driver.

If the call is successful, Line 393 checks if a new TPrint record currently exists. If not doCreatePrintRecord is called to create a new Tprint record (Line 395). If doCreatePrintRecord does not return noErr (Line 396), Line 397 invokes a Stop alert. Line 400 opens the style dialog box.

If the call to PrOpen at Line 388 was not successful, a Caution alert is invoked (Lines 402-403).

Either way, Line 405 closes the Printing Manager (but not the printer driver), releasing the associated memory.

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 415 erases the window preparatory to the simulated document being drawn in the window.

Lines 417-418 establish a rectangle equal to the received page rectangle less 180 pixels in width and 270 pixels in height. (As will be seen, this is the same size as the rectangle to be used in the drawing of each page in the printing graphics port.) Line 419 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 421-423 ensure that the window's font is set to Geneva 10 point.

Line 425 creates a new monostyled edit record with the rectangle established at Lines 417-419 passed in both the destination rectangle parameter and the view rectangle parameter. Line 426 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 428 gets the height of the rectangle established at Lines 417-419. Line 429 calculates how many lines of text will fit into that height. Line 430 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 437), Lines 432-435 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.

doDrawPage

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

Lines 451-452 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 454-455 calculate the number of lines of text that will fit into the height of that rectangle. Line 463 gets the total number of lines in the monostyled edit record created in the function doCalcNumberOfPages.

Lines 458-460 set the printing graphics port's font to Geneva 10 point (the same font and size used to calculate the number of pages).

Line 462 creates a new monostyled edit record with the rectangle established at Lines 451-452 passed in both the destination rectangle parameter and the view rectangle parameter. Line 463 gets a handle to the text in the monostyled edit record created in the function doCalcNumberOfPages.

Line 465 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 466-469 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 472 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 475-482 draw the previously loaded picture at the top left of the rectangle established at Lines 451-452.

Lines 484-486 draw the page number at the bottom left of that rectangle.

doPrintRecordsInfo

doPrintRecordsInfo extracts information from the TPrint, TPrInfo, TPrStyl and TPrJob records and prints it in the window. doDrawRectStrings supports doPrintRecords.

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 586.

doGetPageOrientation

doGetPageOrientation uses PrGeneral to establish the page orientation setting to be used for printing. A TGetRotnBlk structure (Line 601) is used when PrGeneral is used to determine whether landscape orientation has been specified.

After establishing that the current printer driver supports PrGeneral (Line 603), the iopCode field of the TGetRotnBlk structure is assigned the opcode getRtnOp and the hPrint field is assigned the handle to the TPrintRecord (Lines 605-606). PrGeneral is then called (Line 607) with the address of the TGetRotnBlk structure 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 609), otherwise a value representing portrait orientation is returned (Line 611).

If the current printer driver does not support PrGeneral (Line 613), a value representing this situation is returned (Line 614).

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 634), otherwise true is returned (Line 636).

Note that, at Line 631, 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.

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 649) or a Caution alert (Line 653) 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 650).