

15

PRINTING

Includes Demonstration Program Printing

The Printing Manager

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

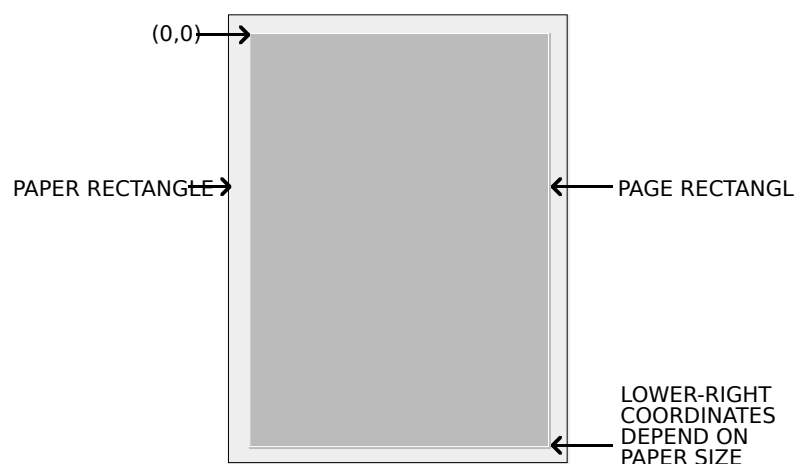


FIG 1 - PAPER AND PAGE RECTANGLES

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 Structure

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.

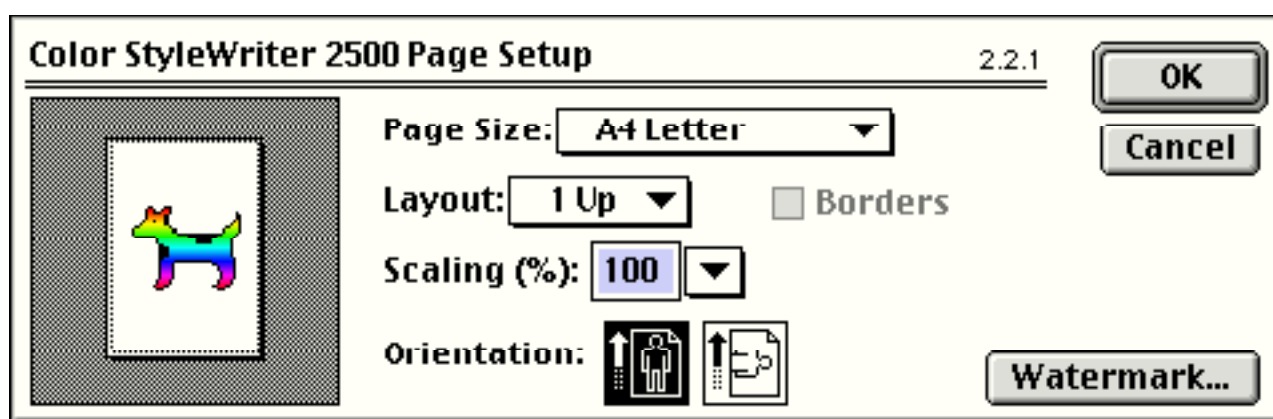


FIG 2 - STYLE DIALOG BOX FOR COLOR STYLEWRITER 2500 PRINTER

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.

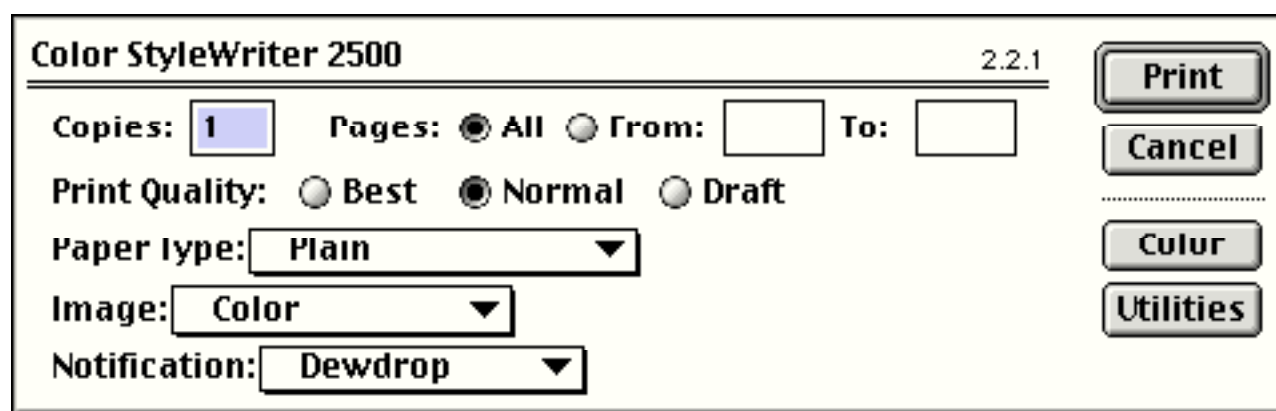


FIG 3 - JOB DIALOG BOX FOR COLOR STYLEWRITER 2500 PRINTER

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` structure (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` structure 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

`PrStDialog` 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 `PrStDialog` because the current printer driver must be open for your application to successfully call `PrStDialog`.

Customised 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 an initialisation function, an item evaluation function and, possibly, an event filter function. A universal procedure pointer to the initialisation function is passed in the second parameter of the function `PrDlgMain`. `PrDlgMain`, not `PrStDialog` and `PrJobDialog`, is used to display a customised style or job dialog box. (See Customising the Style or Job Dialog Box, below.)

The TPrint Structure

To print a document, you need to create a **print structure**. The `TPrint` structure is a data structure of type `TPrint`. Most Printing Manager functions require that you provide a handle to a `TPrint` structure as a parameter.

Your application allocates the memory for a `TPrint` structure itself, using `NewHandle`, and then initialises the `TPrint` structure using `PrintDefault`. Your application may also use an existing

TPrint structure, in which case you can validate the structure using PrValidate. (PrValidate checks all fields of the TPrint structure to ensure compatibility with the current printer.)

When the user chooses the Print... command, your application passes a handle to a TPrint structure 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 structure) of the TPrint structure according to the user's responses.

When the user chooses the Page Setup... command, your application passes a handle to a TPrint structure 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 structure) of the TPrint structure according to the user's responses.

The TPrint structure, including its constituent TPrJob and TPrInfo structures, is shown at Fig 4. Note also the prInfo field (a TPrInfo structure), 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.}
iCopies:integer;  { Number of copies.}
bJDocLoop:Slnt8; { 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:Slnt8;  { Spool file version, set to 0 initially}
bJobX:Slnt8;      { (Reserved)}
end;

TPPrJob = ^TPrJob;

```

```

TPrStl = record
wDev:integer; { Device number of printer.}
iPageV:integer; { (Reserved)}
iPageH:integer; { (Reserved)}
bPort:Slnt8; { (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 RECORDS

The Printing Graphics Port

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

```

type
TPrPortPtr = ^TPrPort;

TPrPort = RECORD
gPort:      GrafPort;  { Printer's graphics port structure. }
gProcs:     QDProcs;   { Functions for printing in the graphics port. }
...
                { More fields for internal use. }

TPPrPort = ^TPrPort;

```

Field Descriptions

- gPort** A graphics port structure, which may be either a CGrafPort or a GrafPort structure, depending on whether the current printer supports colour and greyscale.
- gProcs** A QDProcs structure, which contains pointers to functions which the printer driver may have designated to take the place of QuickDraw functions.

You print text and graphics by drawing into the printing graphics port using QuickDraw drawing functions, just as if you were drawing on the screen. The printer driver installs its own versions of QuickDraw's low-level drawing functions 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 function**.

The TPrJob structure contains a pointer to an idle function in its pIdleProc field (see Fig 4). If this field contains the value NULL, then the printer driver uses its default idle function. The default idle function 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 function 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 function in the pIdleProc field of the TPrJob structure. Your idle function 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 function should also check for clicks in the button and respond accordingly.

If you do not provide your own idle function, 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 function.

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 functions 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 segments² should be unloaded to ensure that the maximum possible memory is available for printing. (This applies to 680x0 code only.)

² See Chapter 23— Miscellany.

- **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).
- **Create or Validate a TPrint Structure.** Use `NewHandle` to allocate storage for a `TPrint` structure, and then initialise that `TPrint` structure using `PrintDefault`. Alternatively, if you are using an existing `TPrint` structure, use `PrValidate` to check that the structure is compatible with the current printer and its driver.
- **Display the Job Dialog Box.** Use `PrJobDialog` to display the job dialog box³ 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` structure. (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 Function (Optional).** If a status dialog box is used, install an idle function in the `pIdleProc` field of the `TPJob` structure 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).⁴
 - **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` functions 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 `TPJob` structure 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

³ The `PrDlgMain` function is used to display a customised job dialog box.

⁴ The value represented by `iPFMaxPgs` is 128.

resources, but it leaves the printer driver open. (The printer driver may be closed using PrDrvClose.)

Creating and Validating the TPrint Structure

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

```
var
  tPrintHdl : THPrint;
  printError : OSErr;
  ...

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
      begin
        DoPrintError(printError);
      end;
    end
  else begin
    { Handle error. }
  end;
```

You can also use an existing TPrint structure (for example, one saved with a document). The following example application-defined routine reads a structure 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 structure is valid for the current version of the Printing Manager and for the current printer driver.

```
function DoGetPrintStructure(refNum : SInt16; tPrintHdl : THPrint;
                           var prRecChanged : boolean) : OSErr;

var
  saveResFile : SInt16;
  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 structure. }
      UseResFile(saveResFile);
      result := PrError;
    end
  else begin
    UseResFile(saveResFile);
    result := kNilHandlePrintErr;
  end;

  DoGetPrintStructure := result;
end;
{ of function DoGetPrintStructure }
```

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 functions 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` structure.

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 a great many existing printer drivers. For this reason, you must be especially wary of, and defensive about, possible error conditions when using Printing Manager functions 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 function, stop printing and call the close function that matches any open function 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 : DocumentRecordHId; displayJobDialog : boolean);
var
  oldPort : GrafPtr;
  numberOfPages, numberOfCopies : SInt16;
  userClickedOK : boolean;
  firstPage, lastPage, copy, page : SInt16;
  tprStatus : TPrStatus;
  printError : SInt16;
begin
  GetPort(oldPort);
```

```

DoUnloadSegments;

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

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

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

        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
              begin
                lastPage := numberOfPages;
              end;

            DoActivateFrontWindow(false, oldPort); { Optional }
            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) then
                      begin
                        if (page <> firstPage) then
                          begin
                            PrCloseDoc(gPrintPortPtr);

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

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

```

```

        end;
    end;
end;

printError := PrError;

PrClose;

if (printError <> noErr) then
begin
    DoPrintError(printError);
end;

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.⁵ 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 function 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 structure 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 routine 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 structure 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 function so specifies, the job dialog box is then displayed. (If the user prints multiple documents at once, the calling function 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 function, the application-defined routine DoJobMerge would, amongst other things, use PrJobMerge to copy data from the first print structure to the print structure 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 structure. 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

⁵ Swapping out code segments applies only to 680x0 code. See Chapter 23 — Miscellany.

TPrJob structure 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 Function (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 function must be installed to handle events in the dialog. The printer driver calls the idle function periodically during the printing process.

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

```

procedure DoPrintIdle;

    var
        oldPort : GrafPtr;
        theEvent : EventRecord;
        gotEvent : boolean;
        itemHit : SInt16;
        handled, cancelled : boolean;

    begin
        GetPort(oldPort);
        SetPort(gPrintStatusDlg);

        gotEvent := WaitNextEvent(everyEvent, theEvent, 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, theEvent, itemHit);

                { DoDidUserCancel should scan for Command-period key-down events (see Chapter 23 -
                  Miscellany) and also for mouse-down events indicating that the user clicked the
                  Stop Printing button. }

                cancelled := DoDidUserCancel;
                if cancelled then
                    begin
                        itemHit := kStopButton;
                    end;

                { 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);
            end;

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

        DoUpdateStatusInformation(cancelled);

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

```

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

- If you draw anything within the idle function, save the printing graphics port upon entry to the idle function and restore it upon exit, as shown in the example.

- If your idle function changes the resource chain⁶, save the reference number of the printer driver's resource file by calling `CurResFile` at the beginning of your idle function. Upon exit, restore the resource chain using `UseResFile`.
- Avoid calling `PrError` within the idle function.

Copies Loop

Before beginning the actual printing process, `printLoop` displays its own status dialog box and installs its own idle function. 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 (or 256, etc.) 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 (or 256, etc.) 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 function `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 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 DolsPrGeneralThere : boolean;
```

⁶ See Chapter 17 — More on Resources.

```
var
  getRotRec : TGetRotnBlk;
  printError : OSErr;

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

  PrGeneral(@getRotRec);

  printError := PrError;
  PrSetError(noErr);

  if (printError = resNotFound) then
    begin
      DolsPrGeneralThere := false;
    end
  else begin
    DolsPrGeneralThere := true;
  end;
end;
{ of function DolsPrGeneralThere }
```

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 : SInt16;

var
  getRotRec : TGetRotnBlk;

begin
  if DolsPrGeneralThere then
    begin
      getRotRec.iOpCode := getRotnOp;
      getRotRec.hPrint := gTPrintHdl;
      PrGeneral(@getRotRec);
      if ((getRotRec.iError = noErr) and (PrError = noErr) and getRotRec.fLandscape) then
        begin
          DoGetPageOrientation := kInLandscapeOrientation;
        end
      else begin
        DoGetPageOrientation := kInPortraitOrientation;
      end
    end
  else begin
    DoGetPageOrientation := kPrGeneralAbsent;
  end;
end;
{ of function DoGetPageOrientation }
```

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 glyphs⁷ 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 function `SetFractEnable` is used to turn fractional glyph widths on and off. `SetFractEnable` affects functions 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.

Customising the Style or Job Dialog Box

As previously stated, 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. Also as previously stated, the function `PrDlgMain`, NOT `PrStDialog` and `PrJobDialog`, is used to display a customised style or job dialog.

Preamble — How the Dialogs Work

The `TPrDlg` structure contains all of the information pertaining to a print dialog and is thus of central importance to the matter of customising style and job dialogs:

```
type
  TPrDlgPtr = ^TPrDlg;

  TPrDlg = RECORD
    Dlg:          DialogRecord;      { A dialog structure. }
```

⁷ A glyph is the visual representation of a character. See Chapter 19 — Text. and TextEdit.

```

pFiltrProc:   ModalFilterUPP;   { The event filter function. }
pItemProc:   PItemUPP;         { The item evaluation function. }
hPrintUsr:   THPrint;          { Handle to a TPrint structure. }
fDolt:       BOOLEAN;          { true means user clicked OK. }
fDone:       BOOLEAN;          { true means user clicked OK or Cancel. }
IUser1:      LONGINT;
IUser2:      LONGINT;
IUser3:      LONGINT;
IUser4:      LONGINT;
END;

```

```
TPPrDlg = ^TPrDlg;
```

Note particularly the fields `pFiltrProc` and `pItemProc`.

When your application calls `PrStdDialog` and `PrJobDialog`, the printer driver calls `PrDlgMain`. `PrDlgMain` is declared as follows:

```
function PrDlgMain(hPrint : THPrint; pDlgInit : PDlgInitUPP) : boolean;
```

`PrDlgMain`, in turn, calls functions which set up the dialog structure, assign a universal procedure pointer to the standard event filter function to the `pFiltrProc` field of the `TPrDlg` structure, and assign a universal procedure pointer to the item evaluation function to the `pItemProc` field. When an item is hit, the function pointed to by the `pItemProc` field is called to handle the hit. When the Print (OK) button is hit, the `THPrint` structure is validated.

Adding Items to the Dialog Box

To customise a print dialog, you must modify the contents of the `TprDlg` structure before the dialog is drawn on the screen. This involves:

- Providing a 'DITL' resource containing the required additional items.
- Defining an **item evaluation function** that handles hits on items in the dialog.
- Defining an **initialisation function** that:
 - Calls `AppendDITL` to append the items to the dialog box.
 - Saves the universal procedure pointer to the driver's item evaluation function in the `pItemProc` field of the `TprDlg` structure. (This will be need to be called from your item evaluation function to handle hits on the dialog's standard items.)
 - Assigns a universal procedure pointer to your item evaluation function in the `pItemProc` field.
 - Returns a pointer to the `TPrDlg` structure.
- If required, defining a custom event filter function and assigning a universal procedure pointer to it to the `pFiltrProc` field of the `TPrDlg` structure.

A universal procedure pointer to the initialisation function should then be passed in the `pDlgInit` parameter of `PrDlgMain`.

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 10 - 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` structure 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 Functions

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

PrGeneral Opcodes

<code>getRslDataOp</code>	= 4	Get resolutions for current printer.
<code>setRslOp</code>	= 5	Set resolutions for a <code>TPrint</code> structure.
<code>draftBitsOp</code>	= 6	Force enhanced draft-quality printing.
<code>noDraftBitsOp</code>	= 7	Cancel enhanced draft-quality printing.
<code>getRotnOp</code>	= 8	Get page orientation of a <code>TPrint</code> structure.
<code>NoSuchRsl</code>	= 1	Resolution not supported.

Data Types

Print Structure

```

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 *TPrint, **THPrint;

```

Printer Information Structure

```

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;
typedef TPrInfo *TPPrInfo;

```

Print Job Structure

```

struct TPrJob
{
    short          iFstPage;        // First page of page range.
    short          iLstPage;        // Last page of page range.
    short          iCopies;         // Number of copies.
    SInt8         bJDocLoop;       // Printing method - draft or deferred.
};

```

Version 2.1

```
    Boolean          fFromUsr;        // (Reserved)
    PIdleUPP         pIdleProc;       // Pointer to an idle function.
    StringPtr       pFileName;       // Spool file name: NULL for default.
    short           iFileVol;        // Spool file volume: set to 0 initially.
    SInt8           bFileVers;      // Spool file version: set to 0 initially.
    SInt8           bJobX;          // (Reserved)
};
typedef struct TPrJob TPrJob;
typedef TPrJob *TPPrJob;
```

Printing Style Structure

```
struct TPrStl
{
    short           wDev;            // Device number of printer.
    short           iPageV;         // (Reserved)
    short           iPageH;         // (Reserved)
    SInt8           bPort;         // (Reserved)
    TFeed          feed;           // Feed type.
};
typedef struct TPrStl TPrStl;
typedef TPrStl *TPPrStl;
```

Printing Graphics Port Structure

```
struct TPrPort
{
    GrafPort       gPort;          // Graphics port for printing.
    QDProcs        gProcs;        // Functions for printing in graphics port.
    long           IGParam1;       // (Reserved)
    long           IGParam2;       // (Reserved)
    long           IGParam3;       // (Reserved)
    long           IGParam4;       // (Reserved)
    Boolean         fOurPtr;        // (Reserved)
    Boolean         fOurBits;      // (Reserved)
};
typedef struct TPrPort TPrPort;
typedef TPrPort *TPPrPort;
```

Printing Status Structure

```
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         flmaging;      // (Reserved)
    THPrint        hPrint;        // Handle to the active printer structure.
    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 Structure

```
struct TPrDlg
{
    DialogRecord   Dlg;           // A dialog structure.
    ModalFilterUPP pFitrProc;     // The event filter function.
    PItemUPP       pItemProc;    // The item evaluation function.
    THPrint        hPrintUsr;    // Handle to a TPrint structure.
    Boolean        fDolt;        // true means user clicked OK.
    Boolean        fDone;       // true means user clicked OK or Cancel.
    long           IUser1;      // (Storage for your application.)
    long           IUser2;      // (Storage for your application.)
    long           IUser3;      // (Storage for your application.)
    long           IUser4;      // (Storage for your application.)
};
typedef struct TPrDlg TPrDlg;
typedef TPrDlg *TPPrDlg;
```

Page Orientation Structure

```

struct TGetRotnBlk
{
    short          iOpCode;          // The getRotnOp opcode.
    short          iError;          // Result code returned by PrGeneral.
    long           lReserved;       // (Reserved)
    THPrint        hPrint;          // Handle to current TPrint structure.
    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);

```

Functions

Opening and Closing the Printing Manager

```

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

```

Initialising and Validating TPrint Structures

```

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);

```

Creating Routine Descriptors

```

#define        NewPDlgInitProc(userRoutine)
                (PDlgInitUPP) NewRoutineDescriptor((ProcPtr)(userRoutine), uppPDlgInitProcInfo,
                GetCurrentArchitecture())
#define        NewPItemProc(userRoutine)
                (PItemUPP) NewRoutineDescriptor((ProcPtr)(userRoutine), uppPItemProcInfo,
                GetCurrentArchitecture())

```

Handling Printing Errors

```

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

```

Demonstration Program

```

{
// PrintingDemo.p
//
// This program:
//
// • Opens a window in which the contents of the main fields of the TPrint, TPrJob,

```

Version 2.1

```
// TPrStl and TPrInfo structures are displayed when the user dismisses the style
// dialog box, and in which the first part of a document is displayed when the user
// hits the Print button in the job dialog box.
//
// • Adds a pop-up menu button, three radio buttons, a checkbox, and a group box to the
// job dialog box.
//
// • Allows the user to print a document containing a picture and text, with the text
// being printed in the font and font size, and with the fractional widths setting,
// specified using the items added to the job dialog box.
//
// The program utilises the following resources:
//
// • 'MBAR' resource and associated 'MENU' resources (preload, non-purgeable).
//
// • A 'WIND' resource (purgeable).
//
// • A 'TEXT' resource (non-purgeable) used for printing.
//
// • A 'PICT' resource (non-purgeable) used for printing.
//
// • 'CNTL' resources (purgeable) for controls added to the job dialog box.
//
// • A 'DITL' resource (purgeable) specifying the items to be appended to the job
// dialog box.
//
// • A 'MENU' resource (preload, non-purgeable) for the pop-up menu button.
//
// }
program PrintingDemo;

{ .....
..... interfaces }

uses

    PrintingUnit, JobDialogAppend;

{ ..... main
program block variables }

var

    mainMenubarHdl : Handle;
    mainMenuHdl : MenuHandle;
    mainEvent : EventRecord;
    mainErr : OSErr;

{ ..... main }

begin

    { ..... initialise
global variables }

    gPrintStructureInited := false;
    gInhibitPrintStructuresInfo := false;

    gWhiteColour.red := $FFFF;
    gWhiteColour.green := $FFFF;
    gWhiteColour.blue := $FFFF;

    gBlackColour.red := $0000;
    gBlackColour.green := $0000;
    gBlackColour.blue := $0000;

    gBlueColour.red := $1818;
    gBlueColour.green := $4B4B;
    gBlueColour.blue := $8181;

{ .....
..... initialise managers }

    DoInitManagers;

    { .....
set up menu bar and menus }
```

```

mainMenuBarHdl := GetNewMBar(rMenuBar);
if (mainMenuBarHdl = nil) then
  begin
    ExitToShell;
  end;
SetMenuBar(mainMenuBarHdl);
DrawMenuBar;

mainMenuHdl := GetMenuHandle(mApple);
if (mainMenuHdl = nil) then
  begin
    ExitToShell;
  end
else begin
  AppendResMenu(mainMenuHdl, 'DRVR');
end;

{ .....
..... open window }

gWindowPtr := GetNewCWindow(rWindow, nil, WindowPtr(-1));
if (gWindowPtr = nil) then
  begin
    ExitToShell;
  end;

SetPort(gWindowPtr);
TextSize(10);
RGBBackColor(gBlueColour);
EraseRect(gWindowPtr^.portRect);

{ ..... load 'TEXT' and
'PICT' resources }

gTextHdl := GetResource('TEXT', rText);
if (gTextHdl = nil) then
  begin
    ExitToShell;
  end;

gPictureHdl := GetPicture(rPicture);
if (gPictureHdl = nil) then
  begin
    ExitToShell;
  end;

{ .....
..... event loop }

gDone := false;

while not gDone do
  begin
    if WaitNextEvent(everyEvent, mainEvent, kMaxLong, nil) then
      begin
        DoEvents(mainEvent);
      end;
  end;

end.
{ of main program }

{ ..... }

{ .....
// PrintingUnit.p
// ..... }

unit PrintingUnit;

interface

{ .....
..... includes }

uses

```

Version 2.1

```
{ Universal Interfaces. }
Appearance, Devices, Fonts, Printing, Processes, Resources, TextUtils, ToolUtils;

{ .....
..... constants }

const

mApple = 128;
mFile = 129;
iQuit = 11;
iPageSetup = 8;
iPrint = 9;
rMenubar = 128;
rWindow = 128;
rText = 128;
rPicture = 128;
rJobDialogAppendDITL = 128;
iPopupButton = 1;
iRadioButton10pt = 2;
iRadioButton12pt = 3;
iRadioButton14pt = 4;
iCheckboxFracWidths = 5;
kMargin = 90;
kMaxLong = $7FFFFFFF;

{ .....
..... global variables }

var

gTPrintHdl : THPrint;
gWindowPtr : WindowPtr;
gDone : boolean;
gPrintStructureInited : boolean;
gInhibitPrintStructuresInfo : boolean;
gEditRecHdl : TEHandle;
gTextHdl : Handle;
gPictureHdl : PicHandle;
gFontNumber : SInt16;
gFontSize : SInt16;
gWhiteColour : RGBColor;
gBlackColour : RGBColor;
gBlueColour : RGBColor;

{ .....
.....routine prototypes }

procedure DoInitManagers;
procedure DoEvents(var theEvent : EventRecord);
procedure DoUpdate(var theEvent : EventRecord);
procedure DoMenuChoice(menuChoice : SInt32);
procedure DoPrinting;
function DoCreatePrintRecord : OSErr;
procedure DoPrStyleDialog;
function DoCalcNumberOfPages(pageRect : Rect) : SInt16;
procedure DoDrawPage(pageRect : Rect; pageNumber : SInt16; numberOfpages : SInt16);
procedure DoDrawPageOrientation;
function DoGetPageOrientation : SInt16;
function DolsPrGeneralThere : boolean;
procedure DoPrintRecordsInfo;
procedure DoDrawRectStrings(s1 : Str255; x1, y1 : SInt16; s2 : Str255; x2, y2 : SInt16;
s3 : Str255);
procedure DoErrorAlert(errorType : SInt16; fatal : boolean);

implementation

uses

{ Other project files. }
JobDialogAppend;

{ ..... DolnitManagers }

procedure DolnitManagers;
var
```



```

numberOfPages, numberOfCopies : SInt16;
firstPage, lastPage, copy, page : SInt16;
printPortPtr : TPPort;
theTPrStatus : TPPrStatusRef;

begin
  GetPort(oldPort);

  PrOpen;

  if (PrError = noErr) then
    begin
      if not gPrintStructureInited then
        begin
          printError := DoCreatePrintRecord;
        end
      else begin
          printError := noErr;
        end;

      gTPrDlgReference := PrJobInit(gTPrintHdl);
      printError := PrError;

      if (printError = noErr) then
        begin
          gInitialisationFunctionUPP := NewPDlgInitProc(@InitialisationFunction);
          gNewItemEvaluateFunctionUPP := NewPItemProc(@ItemEvaluationFunction);
          gEventFilterUPP := NewModalFilterProc(@EventFilter);

          userClickedOK := PrDlgMain(gTPrintHdl, gInitialisationFunctionUPP);

          DisposeRoutineDescriptor(gInitialisationFunctionUPP);
          DisposeRoutineDescriptor(gNewItemEvaluateFunctionUPP);
          DisposeRoutineDescriptor(gEventFilterUPP);

          if userClickedOK then
            begin
              gInhibitPrintStructuresInfo := true;

              numberOfPages := DoCalcNumberOfPages(gTPrintHdl^^.prInfo.rPage);

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

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

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

              for copy := 1 to numberOfCopies do
                begin
                  for page := firstPage to lastPage do
                    begin
                      if ((page - firstPage) mod iPFMaxPgs = 0) then
                        begin
                          if (page <> firstPage) then
                            begin
                              PrCloseDoc(printPortPtr);
                              if ((gTPrintHdl^^.prJob.bJDocLoop = bSpoolLoop) and (PrError = noErr)) then
                                begin
                                  PrPicFile(gTPrintHdl, nil, nil, nil, theTPrStatus);
                                end;
                            end;
                          printPortPtr := PrOpenDoc(gTPrintHdl, nil, nil);
                        end;

                      if (PrError = noErr) then
                        begin
                          PrOpenPage(printPortPtr, nil);
                          if (PrError = noErr) then
                            begin
                              DoDrawPage(gTPrintHdl^^.prInfo.rPage, page, numberOfPages);
                            end;
                          PrClosePage(printPortPtr);
                        end;
                    end;
                end;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```



```

MoveTo(20, 175);
DrawString('First Page:');
NumToString(gTPrintHdl^^.prJob.iFstPage, s2);
MoveTo(190, 175);
DrawString(s2);

```

```

MoveTo(20, 190);
DrawString('Last Page:');
NumToString(gTPrintHdl^^.prJob.iLstPage, s2);
MoveTo(190, 190);
DrawString(s2);

```

```

MoveTo(20, 205);
DrawString('Number of Copies:');
NumToString(gTPrintHdl^^.prJob.iCopies, s2);
MoveTo(190, 205);
DrawString(s2);

```

```

MoveTo(20, 245);
DrawString('Note: Some printer drivers always set the iCopies field of the TPrJob');
MoveTo(20, 260);
DrawString('structure to 1 and handle multiple copies internally. ');
end;

```

```

RGBForeColor(oldForeColour);
RGBBackColor(oldBackColour);
end;
{ of procedure DoPrintRecordsInfo }

```

```
{ } DoDrawRectStrings }
```

```

procedure DoDrawRectStrings(s1 : Str255; x1, y1 : SInt16; s2 : Str255; x2, y2 : SInt16; s3 : Str255);
begin
MoveTo(x1, y1);
DrawString(s1);
MoveTo(x2, y2);
DrawString(' ');
DrawString(s2);
DrawString(' ');
DrawString(s3);
DrawString(' ');
end;
{ of procedure DoDrawRectStrings }

```

```
{ } DoErrorAlert }
```

```

procedure DoErrorAlert(errorType : SInt16; fatal : boolean);
var
paramRec : AlertStdAlertParamRec;
theString, errorString : Str255;
itemHit : SInt16;
ignoredErr : OSErr;

begin
errorString := 'Printing Manager Error';

paramRec.movable := true;
paramRec.helpButton := false;
paramRec.filterProc := nil;
paramRec.defaultText := StringPtr(kAlertDefaultOKText);
paramRec.cancelText := nil;
paramRec.otherText := nil;
paramRec.defaultButton := kAlertStdAlertOKButton;
paramRec.cancelButton := 0;
paramRec.position := kWindowDefaultPosition;

NumToString(SInt32(errorType), theString);
errorString := errorString + theString;

if not fatal then
begin
ignoredErr := StandardAlert(kAlertCautionAlert, @errorString, nil, @paramRec, itemHit);
end
else begin
ignoredErr := StandardAlert(kAlertStopAlert, @errorString, nil, @paramRec, itemHit);
ExitToShell;
end;
end;
{ of procedure DoErrorAlert }

```


- Choose Print... from the File menu, note the items added to the job dialog box, select the desired font, font size, and fractional widths setting using these items, and print 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.

PrintingDemo.p

main program block

After the window is opened, GetResource is called to load the 'TEXT' resource, and GetPicture is used to load the 'PICT' resource, used in the printing demonstration.

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

PrintingUnit.p

Global Variables

gTPrintHdl will be assigned a handle to a TPrint structure. gWindowPtr will be assigned a pointer to the window. gDone controls the exit from the main loop and thus program termination.

gPrintStructureInited will be set to true when a TPrint record has been created and initialised. gInhibitPrintStructuresInfo 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 printing. gPictureHdl will be assigned a handle to the picture used for printing.

gFontNumber will be assigned a font number derived from the font name obtained via user-selection from the Font pop-up menu appended by the application to the job dialog. gFontSize will be assigned a font size obtained via user-selection from the appended radio buttons.

DoUpdate

DoUpdate calls the application-defined routine which draws information in the window. Note that this routine will be called from the application defined event-filter function for the modified job dialog box.

DoMenuChoice

If the user chooses Page Setup... from the File menu, the application-defined function DoPrStyleDialog is called. If the user chooses Print... from the File menu, the application-defined function DoPrinting is called.

DoPrinting

DoPrinting contains the printing loop. It supports printers using deferred printing. However, it does not use a saved TPrint structure (but rather creates one for the print job), and does not use a custom status dialog box and associated idle function. Also, it does not unload unneeded code segments at the beginning. (Unloading code segments applies only to 680x0 code. See Chapter 23 — Miscellany.)

The first line saves the pointer to the current graphics port. The call to PrOpen opens the Printing Manager, together with the current printer driver.

If the global variable gPrintStructureInited indicates that a TPrint structure has not already been created, the application-defined function DoCreatePrintRecord is called to create and initialise a TPrint structure.

Before the printing loop is entered, and because the application will be modifying the standard job dialog box, PrJobInit is called to get a pointer to the TPrDlg structure for the dialog box defined in the resource file for the current printer driver. This pointer will be used in the initialisation function which appends items to the standard job dialog.

If no errors have occurred, routine descriptors are created for the functions associated with the modification of the standard job dialog box, and PrDlgMain is called to display the dialog. A universal procedure pointer to the initialisation function which appends items to the standard job dialog is passed in the last parameter of this call. When the user dismisses the dialog, the routine descriptors are disposed of.

Note that, if an unmodified job dialog is required, the call to PrJobInit, together with the calls to create and dispose of the routine descriptors, should be deleted. In addition, the call to PrDlgMain should be replaced by:

```
userClickedOK := PrJobDialog(gTPrintHdl);
```

Note also that, when the Print button is hit, PrDlgMain calls PrValidate to ensure that the contents of the TPrint structure are compatible with the printer driver for the current printer.

If false is returned by the call to PrDlgMain (that is, the user clicked the Cancel button), the printing loop is bypassed. Otherwise, the first actions are to call an application-defined function to calculate the number of pages, and then retrieve the number of copies, the first page and the last page from the relevant fields of the TPrJob structure.

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 structure which store the first and last page numbers are set back to 1 and iPrPgMax (9999) respectively 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.

The copies loop is then entered, followed by the nested pages loop. The maximum number of pages that can be printed at a time is represented by the constant iPFMaxPgs (128). The first two lines in the pages loop determine whether this is the first or the 129th (or 257th, etc.) time around the pages loop. If, for example, it is the 129th (that is, 128 pages have been printed), PrCloseDoc is called to close the printing graphics port 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 (or 257, etc.) have been printed, PrOpenDoc is called to initialise a new printing graphics port and makes it the current port.

For each page, PrOpenPage re-initialises the printing graphics port, the application-defined function doDrawPage is called to draw that page's contents in the printing graphics port, and PrClosePage wraps up the printing of the current page.

When all pages have been printed, PrCloseDoc closes the printing graphics port and, if the printer driver is using deferred printing, PrPicFile sends the spool file to the printer.

When all copies have been printed (or if control fell through as a result of an error), PrClose releases all memory associated with the Printing Manager (except the printer driver), and the result of the preceding call to PrError is then examined. 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 Caution alert is displayed advising the user of the error and error number.

Finally, the saved graphics port is restored.

DoCreatePrintRecord

DoCreatePrintRecord creates and initialises a TPrint record.

Memory is allocated by the call to NewHandleClear. If the call to allocate memory is successful, PrintDefault initialises the TPrint record to the system standard settings. If this call is successful, the global variable (gPrintStructureInited) which indicates whether an initialised TPrint record exists is set to true. The result of the PrError call is returned to the calling function.

If the call to allocate memory is not successful, the last line simply closes down the program.

DoPrStyleDialog

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

The call to PrOpen opens the Printing Manager and printer driver. If the call is successful, the second if statement determines whether a TPrint record currently exists. If not, DoCreatePrintRecord is called to create a new TPrint record. If DoCreatePrintRecord does not return noErr, a Stop alert is invoked.

PrStylDialog opens the style dialog box.

If the call to PrOpen was not successful, a Caution alert is invoked.

PrClose closes the Printing Manager (but not the printer driver), releasing the associated memory.

DoCalcNumberOfPages

doCalcNumberOfPages is called by DoPrinting 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 structure. TextEdit is not addressed until Chapter 19 — 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 structure 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.

Version 2.1

`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.

`EraseRect` erases the window preparatory to the simulated document being drawn in the window.

The next block establishes 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.) The call to `OffsetRect` 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.

The calls to `TextFont` and `TextSize` set the window's font and font size to those chosen/selected by the user at the pop-up menu and radio buttons added by the application to the standard job dialog box.

`TENew` creates a new monostyled edit record with the previously defined rectangle passed in both the destination rectangle parameter and the view rectangle parameters. `TEInsert` inserts the previously loaded 'TEXT' resource into the edit structure. The `hText` field of the edit structure 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. The first line in this block gets the height of the previously defined rectangle. The next line calculates how many lines of text will fit into that height. The third line 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, the previously loaded picture is drawn 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 structure is retained because it will be used in the following function.

DoDrawPage

`DoDrawPage` is called by `DoPrinting` to draw a specified page in the printing graphics port.

The first block establishes a rectangle equal to the received page rectangle less 180 pixels in width and 270 pixels in height. This smaller rectangle is centred on the page rectangle both laterally and vertically.

The next block calculates the number of lines of text that will fit into the height of that rectangle, and gets the total number of lines in the monostyled edit structure created in the function `DoCalcNumberOfPages`.

The next block sets the printing graphics port's font and font size to the same font and size used to calculate the number of pages.

`TENew` creates a new monostyled edit structure with the previously defined rectangle passed in both the destination rectangle parameter and the view rectangle parameter. The following line gets a handle to the text in the monostyled edit structure created in the function `DoCalcNumberOfPages`.

In the next block, the first line 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. The next four lines 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, `TEInsert` then inserts the text for the page into the newly created edit structure, an action which causes that text to be drawn in the printing graphics port.

If this is the first page, the next block draws the previously loaded picture at the top left of the previously defined rectangle.

The last three lines draw the page number at the bottom left of that rectangle.

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`.

DoGetPageOrientation

`DoGetPageOrientation` uses `PrGeneral` to establish the page orientation setting to be used for printing.

A `TGetRotnBlk` structure is used when `PrGeneral` is used to determine whether landscape orientation has been specified.

If the current printer driver supports `PrGeneral` (determined by the call to `doIsPrGeneralThere`), the `iopCode` field of the `TGetRotnBlk` structure is assigned the opcode `getRtnOp` and the `hPrint` field is assigned the handle to the `TPrintRecord`. `PrGeneral` is then called 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, otherwise a value representing portrait orientation is returned.

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

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, otherwise true is returned.

Note that 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.

DoPrintRecordsInfo and DoDrawRectStrings

DoPrintRecordsInfo extracts information from the TPrint, TPrInfo, TPrStyl and TPrJob structures and prints it in the window if a TPrint structure has been initialised and the global variable gInhibitPrintStructuresInfo contains false. DoDrawRectStrings supports doPrintRecords.

DoErrorAlert

DoErrorAlert is called from DoPrinting and DoPrStyleDialog if an error code is generated following a call to a Printing Manager function. Depending on the nature of the error, either a Stop alert or a Caution alert 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.

JobDialogAppend.p

InitialisationFunction

Recall that, in the function DoPrinting, a universal procedure pointer to InitialisationFunction was passed in the pDlgInit parameter of the PrDlgMain call. PrDialogMain thus calls this function to append the DITL to the end of the job dialog's item list, and to set up the pItemProc (item evaluation) and pFilterProc (event filter function) fields of the TPrDlg structure.

The DITL to be appended contains the following items:

- A pop-up menu button (AddResMenu variant) for font selection. 'FONT' is the specified resource type.
- Three radio buttons for font size selection.
- A checkbox for selecting fractional widths on or off.
- A primary group box (text title variant).

At the first line, the application-defined function DoAppendTheDITL is called to append the items. Amongst other things, DoAppendTheDITL assigns a value to the global variable gFirstAppendedItemNo. This value represents the item number in the job dialog's item list of the first appended item (the pop-up menu button).

The next three blocks make the font pop-up menu WYSIWYG (using essentially the same code as was used in the demonstration program Menus2) check the second radio button and set the global variable holding the current font size to 12pt, and uncheck the checkbox and set fractional widths to off.

The printer driver's item evaluation function will be called upon by the application-defined item evaluation function to handle mouse-downs in the job dialog's standard items. Accordingly, the universal procedure pointer in the TPrDlg record's pItemProc field is saved for future use. The universal procedure pointer to the application-defined item evaluation function is then assigned to the pItemProc field.

As is required by PrDlgMain, the last line returns a pointer to the modified TPrDlg structure.

DoAppendTheDITL

DoAppendTheDITL is called by initialisationFunction to append the new items to the job dialog.

GetResource loads the specified 'DITL' resource and gets a handle to it. CountDITL counts the current number of items in the job dialog. AppendDITL then appends the new items at the bottom of the dialog, causing the dialog box to expand downwards to accommodate them.

Note: If the currently chosen printer is the LaserWriter 8 (driver version 8.4 and later), the result of the AppendDITL call will be somewhat different in that the items will be added to a pane and the name of the application will be inserted into the menu of a pop-up group box. (See Fig 5.)

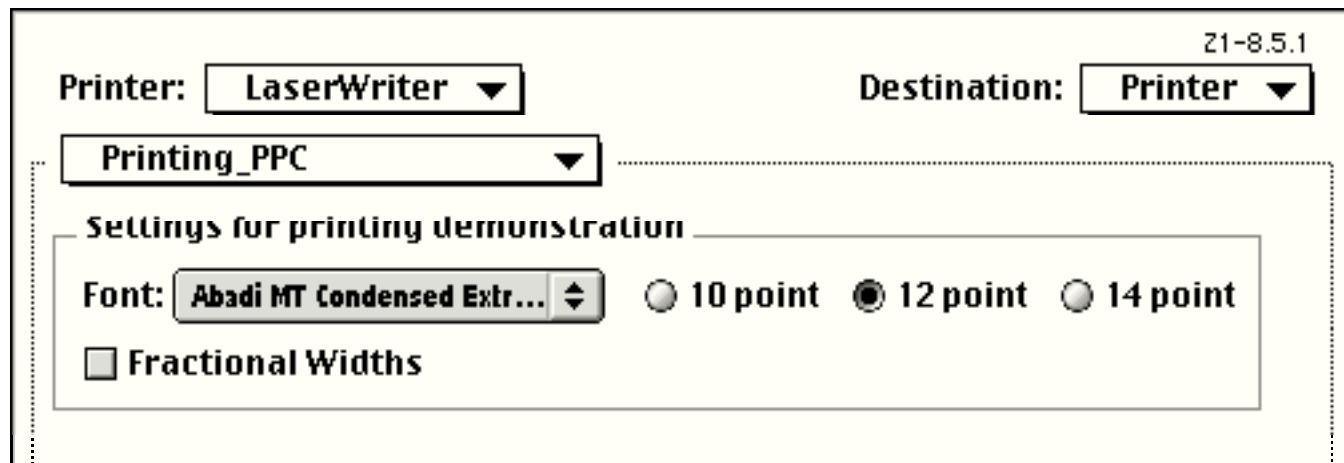


FIG 5 - EFFECT OF AppendDITL - LASERWRITER 8

The global variable `gFirstAppendedItemNo` is then assigned the item number in the new job dialog item list of the first appended item (the pop-up menu button).

ItemEvaluationFunction

`ItemEvaluationFunction` handles item hits in the job dialog. The item number of the item hit is received in the second parameter.

At the first line, the item number of the item hit is "localised". This means that, for example, the localised item number of the pop-up menu button will be 1. In other words, if the localised item number is greater than 0, it will be the item number of one of the appended items; otherwise, it will be the item number of one of the job dialog's standard items.

If the localised item number is greater than 0, and if it is the localised item number for the pop-up menu button, the control's value (that is, the menu item number) is retrieved. `GetMenuItemText` is called to get the text of the menu item, and `GetFNum` is called to get the font number for this font and assign it to the relevant global variable.

If the localised item number is the localised item number for one of the radio buttons, all radio buttons are unchecked, the radio button hit is checked, and the global variable which holds the current text size is assigned the appropriate value. If the localised item number is the localised item number for the checkbox, the current value of that control is flipped and `SetFractEnable` is called to set fractional widths on or off as appropriate.

If the localised item number is 0 or less, the item must be one of the job dialog's standard items. In this case, the printer driver's item evaluation function is called upon to handle the item hit.

EventFilter

`EventFilter` is identical to the custom event filter for modal dialogs introduced at Chapter 8 — Dialogs and Alerts. The use of the event filter is optional. Its use in this program simply allows the dialog, together with windows belonging to background applications, to receive update events.