15

CARBON PRINTING

Demonstration Program: CarbonPrinting

The Carbon Printing Manager

The inclusion of the word "Carbon" in the title of this chapter is quite deliberate, reflecting the fact that, in Carbon, the original **Printing Manager** has been replaced by the new **Carbon Printing Manager**. The Carbon Printing Manager provides a bridge between the fundamentally different Mac OS X and Mac OS 8/9 printing architectures.

When running on Mac OS 8/9, Carbon applications use Classic printer drivers and, generally speaking, Carbon Printing Manager functions simply call through to their original Printing Manager counterparts. When running on Mac OS X, Carbon applications use the new printing architecture and print through different drivers.

The most significant difference between the old Printing Manager and the Carbon Printing Manager is that all data structures are now opaque. Accessor functions are provided to access the information in these objects.

Printing Sessions

In Carbon Printing Manager parlance, an individual printing task is known as a **session**. Carbon applications running on Mac OS X can initiate multiple simultaneous printing sessions, each of which is completely independent of the other. On Mac OS 8/9, printing sessions are also supported, but with the limitation that an application can only ever have one printing session running.

Categories of Carbon Printing Manager Functions

The Carbon Printing Manager API comprises:

- Session functions.
- Non-session functions.
- Universal functions.

The non-session functions do not support simultaneous printing sessions and inherit many of the limitations of the original Printing Manager. For this reason, Apple strongly recommends that the session functions be used instead. (Session and non-session functions must not be mixed within an application.) Accordingly, this chapter addresses the session and universal functions only.

Printer Drivers

Each type of printer has its own **printer driver**. Printer drivers performs the actual printing, translating system software drawing functions as required and send the translated instructions and data to the printer. On Mac OS 8/9, printer drivers are stored in **printer resource files**.

The **current printer** (on Mac OS 8/9, the printer selected by the user in the Chooser) is the printer driver that actually implements the functions defined by the Carbon Printing Manager.

Types and Characteristics of Printer Drivers

In general, there are two types of printer driver:

- QuickDraw printer drivers.
- PostScript printer drivers.

QuickDraw Printers

QuickDraw printers use QuickDraw to render images, which are then sent 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.

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.

Background Printing and Spool Files

Most printer drivers allow users to specify **background printing**, which allows a user to work with an application while documents are printing in the background. On Mac OS 8/9, these printer drivers send printing data to a spool file in the PrintMonitor Documents folder in the System Folder.

Page and Paper Rectangles

Because of an individual printer's mechanical limitations, the printable area of a page is ordinarily less than the physical size of the paper.

Page Rectangle

The **page rectangle** represents the printable area. As shown at Fig 1, the upper-left coordinates of the page rectangle are always (0,0) and the lower-right coordinates represent the maximum printable area height and width for a given printer.

Paper Rectangle

The **paper rectangle** (see Fig 1) represents the physical paper size expressed in the same coordinate system as the page rectangle. The upper left coordinates of the paper rectangle are thus usually negative.

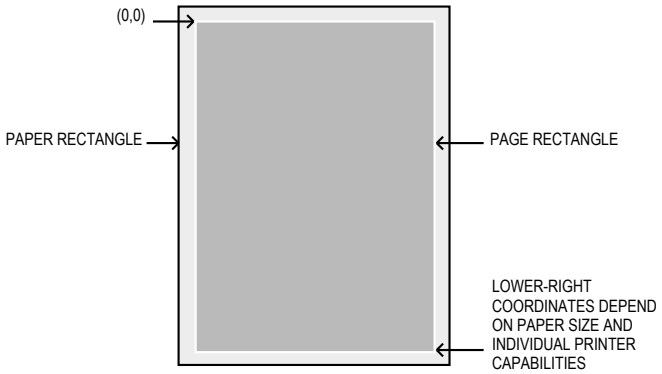
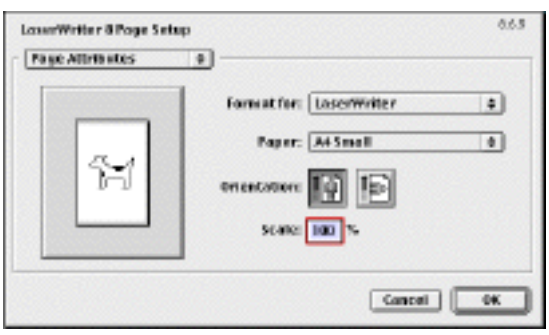


FIG 1 - PAGE AND PAPER RECTANGLES

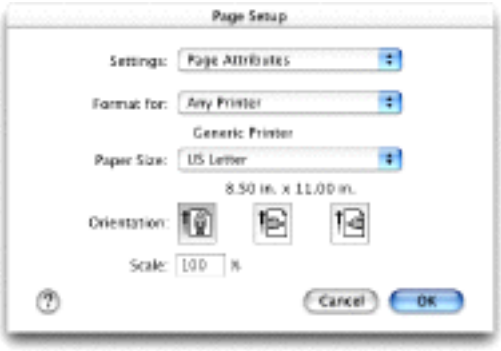
Page Setup Dialogs and Print Dialogs

Page Setup Dialog

In response to the user choosing the **Page Setup...** item from the **File** menu, your application should display the **Page Setup dialog**. For Mac OS 8/9, each printer driver defines its own Page Setup dialog. For Mac OS X, printer manufacturers can extend the standard Page Setup dialog with options specific to their printer. Fig 2 shows the Page Setup dialog.



LASERWRITER 8 PRINTER — MAC OS 8/9



MAC OS X

FIG 2 - PAGE SETUP DIALOG

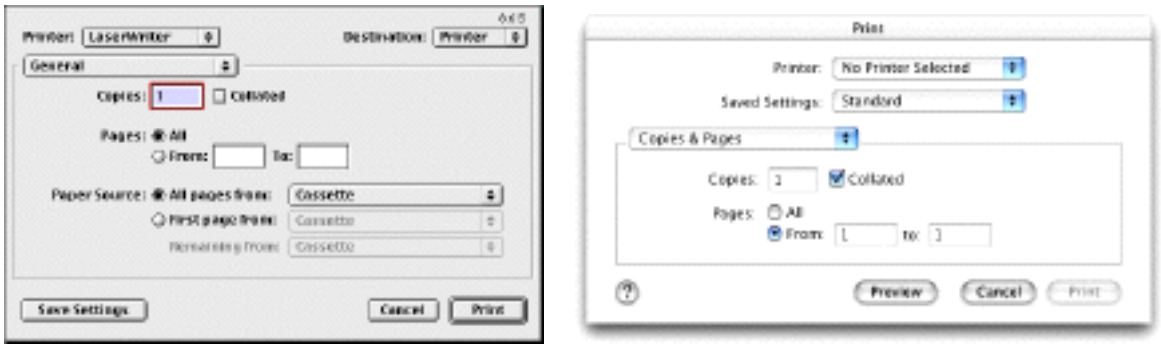
Displaying the Page Setup Dialog and Accessing Settings

`PMSessionPageSetupDialog` is used to display the Page Setup dialog. This function handles all user interaction until the user clicks the **OK** or **Cancel** button.

Settings made in a Page Setup dialog are stored in a `PMPageFormat` object (see below). Your application can use accessor functions to extract information, such as orientation and scaling settings, from this object.

Print Dialog

In response to the user choosing the **Print...** item in the **File** menu, your application should display the **Print dialog**. For Mac OS 8/9, each printer driver defines its own Print dialog. For Mac OS X, printer manufacturers can extend the standard Print dialog with options specific to their printer. Fig 2 shows the Print dialog.



LASERWRITER 8 PRINTER — MAC OS 8/9 MAC OS X

FIG 3 - PRINT DIALOG

Displaying the Print Dialog and Accessing Settings

`PMSessionPrintDialog` is used to display the Print dialog. This function handles all user interaction until the user clicks the **Print** or **Cancel** button.

Settings made in a Print dialog are stored in a `PMPrintSettings` object (see below). Your application can use accessor functions to extract information, such as the page numbers to print, from this object.

Customised Page Setup and Print Dialogs

Many applications add items to the basic Page Setup and Print dialogs so as to provide the user with additional control over printing operations within that application.

If you wish to customise the Page Setup and/or Print dialogs so as to solicit additional information from the user, one option is to use what is sometimes referred to as the AppendDITL method. This requires that you provide an **initialisation function**, an **item evaluation function** and, possibly, an event filter function. A universal procedure pointer to the initialisation function is passed as a parameter in the functions `PMSessionPageSetupDialogMain` and `PMSessionPrintDialogMain`, which are used to display, respectively, customised Page Setup and customised Print dialogs. (See Customising the Page Setup and Print Dialogs, below.)

Preserving the User's Printing Settings

The only information you should preserve each time the user prints the document should be that obtained via the Page Setup dialog. The information supplied by the user through the Print dialog 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.

You can preserve the information obtained via the Page Setup dialog by associating the `PMPageFormat` object with the relevant document's window and by saving it to that document file's data or resource fork when the file is closed. (See Saving and Retrieving a Page Format Object, below.)

Printing Sessions — The PMPrintSession Object

A call to `PMCreateSession` creates a context for printing operations, called a printing session, and initialises a `PMPrintSession` object.

The PMPageFormat and PMPrintSettings Objects

PMPageFormat Object

The `PMPageFormat` object stores information about how the pages of a document should be printed, for example, on what paper size and in what orientation (landscape or portrait).

You use the function PMCreatePageFormat to create an instance of this opaque object. PMCreatePageFormat returns a reference to the object, which is created empty of settings. When your application displays a Page Setup dialog, and the user clicks the **OK** button to dismiss the dialog, the Carbon Printing Manager saves the settings in the PMPageFormat object.

As will be seen, PMPageFormat objects are extensible, meaning that your application can store additional data in them and that your application should never assume that they are of a fixed size.

Accessor Functions

The following describes accessor functions you can use to obtain information contained within PMPageFormat objects.

Function	Description

PMGetAdjustedPaperRect	Gets the paper size.
	On return, the paperRect parameter contains a rectangle describing the size of the paper after scaling, application drawing resolution and orientation settings are applied. The paperRect parameter is of type PMRect:
	`struct PMRect`
	`{`
	`  double top;`
	`  double left;`
	`  double right`
	`  double bottom;`
	`}`
	The size returned is in your application drawing resolution, which you can obtain by calling PMGetResolution.

PMGetAdjustedPageRect	Gets the page size. On return, the pageRect parameter contains a rectangle describing the size of the page after scaling, application drawing resolution, and orientation settings are applied. The pageRect parameter is of type PMRect. The size returned is in your application drawing resolution, which you can obtain by calling PMGetResolution.
PMGetUnadjustedPaperRect	Gets the true size of the paper in points, unaffected by rotation, resolution, or scaling.
PMSetUnadjustedPaperRect	Sets printing to a particular paper size, in points, unaffected by rotation, resolution, or scaling. This function allows applications to request a particular paper size. If the driver cannot handle the specified size an error of kPMValueOutOfRange is returned. If the size is accepted, the application should still call PMGetUnadjustedPaperRect to verify.
PMGetUnadjustedPageRect	Gets the size of the imageable area in points, unaffected by rotation, resolution, or scaling.

PMGetResolution	Gets the application's current drawing resolution.
	On return, the res parameter contains a pointer to a structure of type PMResolution, which describes the resolution at which the Carbon Printing Manager expects your application to render images:
	`struct PMResolution`
	`{`
	`  double hRes;  // The horizontal resolution in dpi.`
	`  double vRes;  // The vertical resolution in dpi.`
	`};`

PMSetResolution	Sets the application's drawing resolution.

PMGetOrientation	Gets the current page orientation setting.
	On return, the orientation parameter contains a pointer to a variable of type PMOrientation, which describes the current page orientation:
	`kPMPortrait         = 1`
	`kPMLandscape        = 2`
	`kPMReversePortrait  = 3`
	`kPMReverseLandscape = 4`

PMSetOrientation	Sets the page orientation for printing.
PMGetScale	Returns the scaling factor currently applied to the page and paper rectangles.
PMSetScale	Sets print scaling.
PMGetPageFormatExtendedData	Gets additional page format data previously stored by your application.
PMSetPageFormatExtendedData	Stores application-supplied data in a PMPageFormat object.

Assigning Default Parameters

A call to PMSessionDefaultPageFormat will assign default parameters to a PMPageFormat object for the specified printing session.

Validating a PMPageFormat Object

A call to PMSessionValidatePageFormat will validate a PMPageFormat object's parameters within the context of the specified printing session. true is returned in the result parameter if any parameters had to be changed.

PMPrintSettings Object

The PMPrintSettings object stores information such as the number of copies, range of pages to print, etc., for a particular printing session.

You use the function PMCreatePrintSettings to create an instance of this opaque object. PMCreatePrintSettings returns a reference to the object, which is created empty of settings. When your application displays a Print dialog, and the user clicks the **Print** button to dismiss the dialog, the Carbon Printing Manager saves the settings in the PMPrintSettings object.

As will be seen, PMPrintSettings objects are extensible, meaning that your applications can store additional data in them and that your application should never assume that they are of a fixed size.

Accessor Functions

The following describes the accessor functions you can use to obtain information contained within PMPrintSettings objects.

Function	Description
PMGetFirstPage	Gets the starting page number of the pages to be printed.
PMSetFirstPage	Sets the page number of the first page to be printed.
PMGetLastPage	Gets the last page number of the pages to be printed.
PMSetLastPage	Sets the page number of the first page to be printed.
PMGetPageRange	Gets the valid range of pages to print, as previously set by PMSetPageRange. If none was set by the application, the default range (1-32000) is returned.
PMSetPageRange	Sets the valid range of pages to be printed. On Mac OS X, if the user enters a value outside this range in the Print dialog, an alert message is displayed. This feature is not supported on Mac OS 8/9. The relationship between the page range set by the application using PMSetPageRange and the first and last pages set by the user in the Print dialog (and retrieved by PMGetFirstPage and PMGetLastPage) is shown at Fig 4.
PMGetCopies	Gets the number of copies to be printed.
PMSetCopies	Sets the default value to be displayed for the number of copies to be printed.
PMGetPrintSettingsExtendedData	Gets additional print settings previously saved by your application.
PMSetPrintSettingsExtendedData	Stores application-supplied data in a PMPrintSettings object.

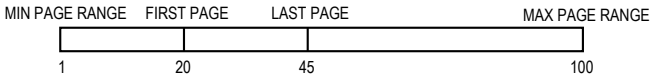


FIG 4 - FIRST PAGE, LAST PAGE, AND PAGE RANGE (EXAMPLE)

Note: You pass true in the lock parameter of the functions PMSetCopies, PMSetFirstPage, and PMSetLastPage if you wish to lock, respectively, the number-of-copies, first-page, and last-page fields in the Print dialog. Passing true only affects printer drivers for Mac OS X and LaserWriter printer drivers version 8.7 and later. If you pass true for other printer drivers, these functions will return kPMLockIgnored.

Assigning Default Parameters

A call to PMSessionDefaultPrintSettings will assign default parameters to a PMPrintSettings object for the specified printing session.

Validating a PMPrintSettings Object

A call to `PMSessionValidatePrintSettings` will validate a `PMPrintSettings` object's parameters within the context of the specified printing session. `true` is returned in the `result` parameter if any parameters had to be changed.

Printing a Document

The following describes a typical approach to printing a document.

When the User Chooses Page Setup... From the File Menu

When the user invokes the Page Setup dialog:

- Call a function which:
  - Calls `PMCreateSession` to create a `PMPrintSession` object, and associates that object with the document's window.
  - If a `PMPageFormat` object has not previously been created and associated with the document's window:
    - Calls `PMCreatePageFormat` to create a `PMPageFormat` object.
    - Calls `PMSessionDefaultPageFormat` to assign default parameters to the `PMPageFormat` object.
    - Associates the `PMPageFormat` object with the document's window.
  - If a `PMPageFormat` object has previously been created and associated with the document's window, calls `PMSessionValidatePageFormat` to validate the `PMPageFormat` object's parameters within the context of this printing session.
- Call `PMSessionPageSetupDialog` to present the Page Setup dialog, handle all user interaction within the dialog, and record the user's settings in the `PMPageFormat` object.
- When the user dismisses the Page Setup dialog, call `PMRelease` to release the `PMPrintSession` object.

When the User Chooses Print... From the File Menu

When the user invokes the Print dialog:

- As when the Page Setup dialog is invoked (see above), call a function which:
  - Creates a `PMPrintSession` object and associates it with the document's window.
  - Either creates a `PMPageFormat` object, assigns default parameters to it, and associates it with the document's window or, if this has previously been done, validates the existing `PMPageFormat` object's parameters.
- Call `PMCreatePrintSettings` to create a `PMPrintSettings` object.
- Call `PMSessionDefaultPrintSettings` to assign default parameters to the `PMPrintSettings` object.
- Associate the `PMPrintSettings` object with the document's window.
- For Mac OS X only:
  - Call `PMSetPageRange` to set the valid range of pages that can be printed.
  - Call `PMSetFirstPage` and `PMSetLastPage` to set the default first and last pages to be printed, as displayed in the **From** and **To** fields of the Print dialog. (To clear the **From** and **To** fields and select the **All** radio button, pass `kPMPrintAllPages` in the `PMSetLastPage` call.)
- Call `PMSessionPrintDialog` to present the Print dialog, handle all user interaction within the dialog, and record the user's settings in the `PMPrintSettings` object.

- If the user clicks the **Cancel** button in the Print dialog, call `PMRelease` to release the `PMPrintSettings` and `PMPrintSession` objects, and disassociate them from the document's window.
- If the user clicks the **Print** button in the Print dialog, call your application's **printing loop** function.

The Printing Loop Function

The printing loop is the part of your application's code that performs the actual printing. The function containing the printing loop should perform the following actions:

- Call `PMGetFirstPage` and `PMGetLastPage` to get the first and last pages to print, as entered by the user in the Print dialog.
- Call a function which calculates the actual number of pages in the document. If necessary, adjust the value in the variable containing the last page as returned by `PMGetLastPage`.
- For Mac OS X, call `PMSetFirstPage` and `PMSetLastPage` to tell the Carbon Printing Manager which pages will be spooled so that the progress dialog can display an accurate page count. Pass in the values returned by the `PMGetFirstPage` and `PMGetLastPage` calls, the latter adjusted as necessary.
- Call `PMSessionBeginDocument` to create an instance of a `PMPrintContext` object and thus establish a graphics context for imaging the document. (As will be seen, a function exists to obtain the graphics port (that is, the printing port) associated with this object.)
- Print the pages. In the page-printing loop, all of the pages in the document should be spooled and the Carbon Printing Manager relied upon to print the correct page range as specified in the Print dialog. Note that your printing loop does not have to concern itself with the number of copies, since this is handled automatically by the Carbon Printing Manager. The pages loop should:
  - Call `PMSessionBeginPage` to initialise the printing port. (Note that, for Mac OS 8/9 only, you can pass a scaling (bounding) rectangle in the `pageFrame` parameter, in which case all items drawn in the printing port will be scaled to fit this page rectangle. If no scaling is required, pass `NULL` in the `pageFrame` parameter.)
  - Call `GetPort` to save the current graphics port, call `PMSessionGetGraphicsContext` to obtain the current printing port, and call `SetPort` to set that port as the current port.
  - Call a function which draws the relevant page in the printing port, then call `SetPort` to set the port saved by the `GetPort` call as the current port.
  - Call `PMSessionEndPage` to print the page.
- When all copies of all pages have been printed, call `PMSessionEndDocument` to close the printing graphics port.
- Call `PMRelease` to release the `PMPrintSettings` and `PMPrintSession` objects.

Call Sequence And Scope

When writing functions which call Carbon Printing Manager functions, bear in mind that the Carbon Printing Manager enforces a **sequence** of steps in the printing loop, and defines a valid **scope** for each Carbon Printing manager function. Functions used out of sequence will return a result code of `kPMOutOfScope`.

Sequence and Scope: Session Functions

The following, in which scope level is represented by indentation, shows calling sequence and scope requirements of the main session printing functions. It shows, for example, that you can call `PMSessionGetGraphicsContext` only after calling `PMSessionBeginPage`.

```
PMCreateSession
   PMSessionDefaultPageFormat
   PMSessionValidatePageFormat
   PMSessionDefaultPrintSettings
   PMSessionValidatePrintSettings
   PMSessionError

   PMSessionPageSetupDialog
   PMSessionPrintDialog
   PMSessionPageSetupDialogInit              These functions must be called
   PMSessionPrintDialogInit                  before PMSessionBeginDocument
   PMSessionPrintDialogMain
   PMSessionPageSetupDialogMain

   PMSessionBeginDocument
      PMSessionBeginPage
         PMSessionGetGraphicsContext
      PMSessionEndPage
   PMSessionEndDocument
```

In general, functions may be called in any order with respect to other functions at the same or lower scope level.

Sequence and Scope: Universal Functions

The following are the main universal functions which have no calling sequence or scope, and which may generally be used anywhere in your printing code:

```
PMCreatePageFormat          PMGetLastPage               PMGetScale
PMFlattenPageFormat         PMSetLastPage               PMSetScale
PMUnflattenPageFormat       PMGetCopies                 PMGetResolution
PMCreatePrintSettings       PMSetCopies                 PMSetResolution
PMFlattenPrintSettings      PMGetAdjustedPaperRect      PMGetPageFormatExtendedData
PMUnflattenPrintSettings    PMGetAdjustedPageRect       PMSetPageFormatExtendedData
PMGetPageRange              PMGetUnadjustedPageRect     PMGetPrintSettingsExtendedData
PMSetPageRange              PMSetUnadjustedPaperRect    PMSetPrintSettingsExtendedData
PMGetFirstPage              PMGetOrientation
PMSetFirstPage              PMSetOrientation
```

Handling Printing Errors

The Carbon Printing Manager must necessarily bear the heavy burden of maintaining backward compatibility with early 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 Carbon Printing Manager functions.

Do not display an alert to report an error until the end of the printing loop. This is important for two reasons:

- If you display an alert 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 alert reporting the error.

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 may 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[1] 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 Page Setup and Print Dialogs

As previously stated, you may want to add additional options to the Page Setup and Print dialogs so that the user can further customise the printing process. For example, you might want to add a "skip blank pages" checkbox to a Print dialog.

> ***Note***
>
> On Mac OS X, the **printing dialog extension** (PDE) mechanism may be used to extend the Page Setup and Print dialogs. The PDE mechanism provides great flexibility in extending printing dialogs. However, because the PDE mechanism applies only to Mac OS X, that method of extending Page Setup and Print dialogs is not addressed in this book. The method addressed is sometimes referred to as the AppendDITL method.
>
> A limitation of the AppendDITL method is that it prevents you from creating the Page Setup and Print dialogs as window-modal (sheet) dialogs. This limitation does not apply when the PDE method is used.

The functions `PMSessionPageSetupDialogMain` and `PMSessionPrintDialogMain` are used to display Page Setup and Print dialogs customised using the AppendDITL method.

The PMDialog Object

Your application uses a reference to a `PMDialog` object when creating custom Page Setup and Print dialogs. The functions `PMSessionPageSetupDialogInit` and `PMSessionPrintDialogInit` functions are used to create and initialise instances of this opaque object. Taking the reference to this object as a parameter, the function `PMGetDialogPtr` returns a pointer to the dialog structure.

[1] A glyph is the visual representation of a character. See Chapter 21.

Customising a Print Dialog

As an example, to customise a Print dialog, you must modify the contents of the PMDialog object 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 events involving the additional items.
- Defining and installing an **initialisation function** that:
  - Calls AppendDITL to append the additional items to the dialog.
  - Using PMGetItemProc, gets the universal procedure pointer to the printer driver's item evaluation function from the PMDialog object, and saves it. (The printer driver's item evaluation function will need to be called from your item evaluation function to handle hits on the dialog's standard items.)
  - Calls PMSetItemProc to set a universal procedure pointer to your item evaluation function in the PMDialog object.
- If required, defining a custom event filter function and setting a universal procedure pointer to it in the PMDialog object using PMSetModalFilterProc.

A universal procedure pointer to the initialisation function should then be passed in the myInitProc parameter of PMSessionPrintDialogMain, which displays the customised Print dialog.

Displaying Page Setup and Print Dialogs as Window-Modal (Sheet) Dialogs

To cause Page Setup and Print dialogs to be created as window-modal (sheet) dialogs on Mac OS X, you should call the function PMSessionUseSheets immediately before the calls to PMSessionPageSetupDialog and PMSessionPrintDialog, passing the window reference for the parent window in the documentWindow parameter and a universal procedure pointer to an application-defined (callback) function in the sheetDoneProc parameter. The callback function should perform the actions required immediately following dismissal of the dialog, and should be declared like this:

```
void  myPageSetupSheetDoneFunction(PMPrintSession printSession, WindowRef documentWindow,
                                   Boolean accepted);
```

The accepted formal parameter will be set to true if the **Print**/**OK** push button is clicked and to false if the **Cancel** push button is clicked.

The functions NewPMSheetDoneUPP and DisposePMSheetDoneUPP create and dispose of the universal procedure pointers.

Saving and Retrieving a Page Format Object

As previously stated, the only information you should preserve each time the user prints a document should be that obtained via the Page Setup dialog. Ordinarily, therefore, you will want your application to save the flattened PMPageFormat object associated with a specific document in either the data fork or the resource fork of that document's file when you save the document itself.

You can store additional data inside the PMPageFormat object before it is flattened and saved by calling PMSetPageFormatExtendedData, whose parameters include a unique code identifying the data, the size of the data, and a pointer to the data. When the flattened PMPageFormat object is retrieved and unflattened, you can obtain this data by calling PMGetPageFormatExtendedData.

Saving a flattened PMPageFormat object to the resource fork of a file, and retrieving the flattened object from the file, is demonstrated in the demonstration program associated with Chapter 19.

Printing From the Finder — Mac OS 8/9

Users generally print documents that are open on the screen one at a time while the application that created the document is running. However, on Mac OS 8/9, 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:

- Use saved or default page setup settings instead of displaying the Page Setup dialog.
- Display the Print dialog once only, and use `PMCopyPrintSettings` to apply the information specified by the user to all of the selected documents.
- Remain open unless and until the Finder sends it a Quit Application event.

Main Carbon Printing Manager Constants, Data Types and Functions

Constants

Unwanted Data

```
KPMNoData               = NULL
KPMDontWantSize         = NULL
KPMDontWantData         = NULL
KPMDontWantBoolean      = NULL
KPMNoPrintSettings      = NULL
kPMNoPageFormat         = NULL
kPMNoReference          = NULL
```

Page Orientation

```
kPMPortrait             = 1
kPMLandscape            = 2
kPMReversePortrait      = 3
kPMReverseLandscape     = 4
```

User Cancelled

```
kPMCancel               = 128
```

For PMSetPageRange and PMSetLastPage

```
kPMPrintAllPages        = -1
```

Result Codes

```
kPMNoError              = 0
kPMGeneralError         = -30870
kPMOutOfScope           = -30871
kPMInvalidParameter     = paramErr
kPMNoDefaultPrinter     = -30872
kPMNotImplemented       = -30873
kPMNoSuchEntry          = -30874
kPMInvalidPrintSettings = -30875
kPMInvalidPageFormat    = -30876
kPMValueOutOfRange      = -30877
kPMLockIgnored          = -30878
kPMInvalidPrintSession  = -30879
kPMInvalidPrinter       = -30880
kPMObjectInUse          = -30881
```

Data Types

Opaque Types

```
typedef struct OpaquePMPrintSession*  PMPrintSession;
typedef struct OpaquePMPageFormat*    PMPageFormat;
typedef struct OpaquePMPrintSettings* PMPrintSettings;
typedef struct OpaquePMPrintContext*  PMPrintContext;
typedef struct OpaquePMDialog*        PMDialog;
```

PMRect

```
struct PMRect
{
  double top;
  double left;
  double right;
  double bottom;
}
```

PMResolution

```
struct PMResolution
{
  double hRes;
  double vRes;
}
```

Functions

Managing Printing Objects

```
OSStatus  PMRetain(PMObject object);
OSStatus  PMRelease(PMObject object);
```

Print Loop

```
OSStatus  PMCreateSession(PMPrintSession *printSession);
OSStatus  PMSessionBeginDocument(PMPrintSession printSession,PMPrintSettings printSettings,
          PMPageFormat pageFormat);
OSStatus  PMSessionEndDocument(PMPrintSession printSession);
OSStatus  PMSessionBeginPage(PMPrintSession printSession,PMPageFormat pageFormat,
          const PMRect *pageFrame);
OSStatus  PMSessionEndPage(PMPrintSession printSession);
OSStatus  PMSessionGetGraphicsContext(PMPrintSession printSession,
          CFStringRef graphicsContextType,void **graphicsContext);
```

Page Format and Print Settings Objects

```
OSStatus  PMCreatePageFormat(PMPageFormat *pageFormat);
OSStatus  PMCreatePrintSettings(PMPrintSettings *printSettings);
OSStatus  PMSessionDefaultPageFormat(PMPrintSession printSession,PMPageFormat pageFormat);
OSStatus  PMSessionDefaultPrintSettings(PMPrintSession printSession,
          PMPrintSettings printSettings);
OSStatus  PMSessionValidatePageFormat(PMPrintSession printSession,PMPageFormat pageFormat,
          Boolean *result);
OSStatus  PMSessionValidatePrintSettings(PMPrintSession printSession,
          PMPrintSettings printSettings,Boolean *result);
OSStatus  PMCopyPageFormat(PMPageFormat formatSrc,PMPageFormat formatDest);
OSStatus  PMCopyPrintSettings(PMPrintSettings settingSrc,PMPrintSettings settingDest);
OSStatus  PMFlattenPageFormat(PMPageFormat pageFormat,Handle *flatFormat);
OSStatus  PMUnflattenPageFormat(Handle flatFormat,PMPageFormat *pageFormat);
OSStatus  PMFlattenPrintSettings(PMPrintSettings printSettings,Handle *flatSetting);
OSStatus  PMUnflattenPrintSettings(Handle flatSetting,PMPrintSettings *printSettings);
```

Displaying the Page Setup and Print Dialogs

```
OSStatus  PMSessionPageSetupDialog(PMPrintSession printSession,PMPageFormat pageFormat,
          Boolean *accepted);
OSStatus  PMSessionPrintDialog(PMPrintSession printSession,PMPrintSettings printSettings,
          PMPageFormat constPageFormat,Boolean* accepted);
OSStatus  PMSessionUseSheets(PMPrintSession printSession,WindowRef documentWindow,
          PMSheetDoneUPP sheetDoneProc);
```

Customising Page Setup and Print Dialogs

```
OSStatus  PMSessionPageSetupDialogInit(PMPrintSession printSession,PMPageFormat pageFormat,
          PMDialog *newDialog);
OSStatus  PMSessionPrintDialogInit(PMPrintSession printSession,PMPrintSettings printSettings,
          PMPageFormat constPageFormat,PMDialog *newDialog);
OSStatus  PMSessionPageSetupDialogMain(PMPrintSession printSession,PMPageFormat pageFormat,
          Boolean *accepted,PMPageSetupDialogInitUPP myInitProc);
OSStatus  PMSessionPrintDialogMain(PMPrintSession printSession,PMPrintSettings printSettings,
          PMPageFormat constPageFormat,Boolean *accepted,PMPrintDialogInitUPP myInitProc);
OSStatus  PMSetModalFilterProc(PMDialog pmDialog,ModalFilterUPP filterProc);
OSStatus  PMSetItemProc(PMDialog pmDialog,PMItemUPP itemProc);
OSStatus  PMGetItemProc(PMDialog pmDialog,PMItemUPP *itemProc);
OSStatus  PMGetDialogPtr (PMDialog pmDialog, DialogRef *theDialog);
```

Getting and Setting Page Setup Information

```
OSStatus  PMGetAdjustedPaperRect(PMPageFormat pageFormat,PMRect *paperRect);
OSStatus  PMGetAdjustedPageRect(PMPageFormat pageFormat,PMRect *pageRect);
```

```
OSStatus  PMGetUnadjustedPaperRect(PMPageFormat pageFormat,PMRect *paperSize);
OSStatus  PMSetUnadjustedPaperRect(PMPageFormat pageFormat,const PMRect paperSize);
OSStatus  PMGetUnadjustedPageRect(PMPageFormat pageFormat,PMRect *pageSize);
OSStatus  PMGetResolution(PMPageFormat pageFormat,PMResolution *res);
OSStatus  PMSetResolution(PMPageFormat pageFormat,const PMResolution res);
OSStatus  PMSetOrientation(PMPageFormat pageFormat,PMOrientation orientation,Boolean lock);
OSStatus  PMGetOrientation(PMPageFormat pageFormat,PMOrientation *orientation);
OSStatus  PMGetScale(PMPageFormat pageFormat,double *scale);
OSStatus  PMSetScale(PMPageFormat pageFormat,double scale);
OSStatus  PMGetPageFormatExtendedData(PMPageFormat pageFormat,OSType dataID,UInt32 *size,
          void *extendedData);
OSStatus  PMSetPageFormatExtendedData(PMPageFormat pageFormat,OSType dataID,UInt32 size,
          void *extendedData);
```

Getting and Setting Printing Information

```
OSStatus  PMGetFirstPage(PMPrintSettings printSettings,UInt32 *first);
OSStatus  PMSetFirstPage(PMPrintSettings printSettings,UInt32 first,Boolean lock);
OSStatus  PMGetLastPage(PMPrintSettings printSettings,UInt32 *last);
OSStatus  PMSetLastPage(PMPrintSettings printSettings,UInt32 last,Boolean lock);
OSStatus  PMGetPageRange(PMPrintSettings printSettings,UInt32 *minPage,UInt32 *maxPage);
OSStatus  PMSetPageRange(PMPrintSettings printSettings,UInt32 minPage,UInt32 maxPage);
OSStatus  PMGetCopies(PMPrintSettings printSettings,UInt32 *copies);
OSStatus  PMSetCopies(PMPrintSettings printSettings,UInt32 copies,Boolean lock);
OSStatus  PMGetPrintSettingsExtendedData(PMPrintSettings printSettings,OSType dataID,
          UInt32 *size,void *extendedData);
OSStatus  PMSetPrintSettingsExtendedData(PMPrintSettings printSettings,OSType dataID,
          UInt32 size,void *extendedData);
```

Creating and Disposing of Universal Procedure Pointers

```
PMPageSetupDialogInitUPP NewPMPageSetupDialogInitUPP(PMPageSetupDialogInitProcPtr userRoutine);
PMPrintDialogInitUPP     NewPMPrintDialogInitUPP(PMPrintDialogInitProcPtr userRoutine);
PMItemUPP                NewPMItemUPP(PMItemProcPtr userRoutine);
PMIdleUPP                NewPMIdleUPP(PMIdleProcPtr userRoutine);
PMSheetDoneUPP           NewPMSheetDoneUPP(PMSheetDoneProcPtr userRoutine);
void      DisposePMPageSetupDialogInitUPP(PMPageSetupDialogInitUPP userUPP);
Void      DisposePMPrintDialogInitUPP(PMPrintDialogInitUPP userUPP);
void      DisposePMItemUPP(PMItemUPP userUPP);
void      DisposePMIdleUPP(PMIdleUPP userUPP);
void      DisposePMSheetDoneUPP(PMSheetDoneUPP userUPP);
```

Errors

```
OSStatus  PMSessionError(PMPrintSession printSession);
```

Application-Defined (Callback) Functions

```
void      myPMDialogSheetDoneFunction(PMPrintSession printSession, WindowRef documentWindow,
                                      Boolean accepted);
```

Demonstration Program CarbonPrinting Listing

```
// ****************************************************************************************
// CarbonPrinting.h                                                        CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program:
//
// •  Demonstrates printing using the Carbon Printing Manager session functions.
//
// •  Opens two windows. The first window is a document window in which is displayed the
//    document to be printed.  The second window displays some printing-related information
//    obtained from PMPageFormat and PMPrintSettings objects.
//
// •  Customises the Pring dialog by adding a pop-up menu button, three radio buttons, a
//    checkbox, and a group 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 Print dialog.
//
// The customising of the Print dialog uses the AppendDITL method.  Accordingly, on Mac OS X,
// the dialogs are application-modal and are not displayed as window-modal sheet dialogs.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  'MBAR' resource and associated 'MENU' resources (preload, non-purgeable).
//
// •  Two 'WIND' resources (purgeable).
//
// •  A 'TEXT' resource (purgeable) used for printing.
//
// •  A 'PICT' resource (non-purgeable) used for printing.
//
// •  'CNTL' resources (purgeable) for controls added to the Print dialog box.
//
// •  A 'DITL' resource (purgeable) specifying the items to be appended to the Print dialog
//    box.
//
// •  A 'MENU' resource (preload, non-purgeable) for the pop-up menu button.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ..... includes

#include <Carbon.h>

// ..... defines

#define rMenubar               128
#define mAppleApplication      128
#define  iAbout                1
#define mFile                  129
#define  iPageSetup            9
#define  iPrint                10
#define  iQuit                 12
#define mFont                  131
#define rDocWindow             128
#define rInfoWindow            129
#define rText                  128
#define rPicture               128
#define rPrintDialogAppendDITL 128
#define  iPopupButton          1
```

```
#define  iRadioButton10pt      2
#define  iRadioButton12pt      3
#define  iRadioButton14pt      4
#define  iCheckboxFracWidths   5
#define kMargin                90
#define MAX_UINT32             0xFFFFFFFF
#define MIN(a,b)               ((a) < (b) ? (a) : (b))

// ..... typedefs

typedef struct
{
  PMPrintSession  printSession;
  PMPageFormat    pageFormat;
  PMPrintSettings printSettings;
  TEHandle        editTextStrucHdl;
  PicHandle       pictureHdl;
} docStructure, *docStructurePtr, **docStructureHdl;

// ..... function prototypes

void      main                          (void);
void      doPreliminaries               (void);
OSErr      quitAppEventHandler          (AppleEvent *,AppleEvent *,SInt32);
void      doGetDocument                 (void);
void      doEvents                      (EventRecord *);
void      doUpdate                      (EventRecord *);
void      doUpdateDocumentWindow        (WindowRef);
void      doMenuChoice                  (SInt32);
OSStatus  doCreateOrValidatePageFormat  (WindowRef);
OSStatus  doPageSetUpDialog             (WindowRef);
OSStatus  doPrintSettingsDialog         (WindowRef);
OSStatus  doPrinting                    (WindowRef);
SInt16    doCalcNumberOfPages           (WindowRef,Rect);
void      doDrawPage                    (WindowRef,Rect,SInt16,SInt16);
void      doDrawPrintInfo               (void);
void      doDrawRectStrings             (Str255,SInt16,SInt16,Str255,SInt16,SInt16,Str255);
void      doErrorAlert                  (OSStatus);
void      doConcatPStrings              (Str255,Str255);

void      initialisationFunction        (PMPrintSettings, PMDialog *);
void      itemEvaluationFunction        (DialogPtr,SInt16);
Boolean   eventFilter                   (DialogPtr,EventRecord *,SInt16 *);

// ********************************************************************************************
// CarbonPrinting.c
// ********************************************************************************************

// ..... includes

#include "CarbonPrinting.h"

// ..... global variables

Boolean         gRunningOnX = false;
WindowRef       gDocumentWindowRef, gPrintInfoWindowRef;
SInt16          gFontNumber;
SInt16          gFontSize;
Boolean         gDone;
PMItemUPP       gNewItemEvaluateFunctionUPP;
ModalFilterUPP  gEventFilterUPP;
PMPrintSettings gPrintSettings = kPMNoPrintSettings;
PMDialog        gPMDialog;
UInt32          gFirstPage, gLastPage, gCopies;
Boolean         gDrawPrintSettingsStuff;

// ********************************************************************************** main

void  main(void)
```

```
{
  MenuBarHandle   menubarHdl;
  SInt32          response;
  MenuRef         menuRef;
  docStructureHdl docStrucHdl;
  SInt16          fontNum;
  RGBColor        whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  RGBColor        blueColour  = { 0x4444, 0x4444, 0x9999 };
  Rect            portRect;
  EventRecord     eventStructure;
  Boolean         gotEvent;

  // ..... do prelimiaries

  doPreliminaries();

  // ..... set up menu bar and menus

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

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }

    gRunningOnX = true;
  }

  // ..... open document window and attach document structure

  if(!(gDocumentWindowRef = GetNewCWindow(rDocWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gDocumentWindowRef);
  GetFNum("\pGeneva",&fontNum);
  TextFont(fontNum);
  TextSize(10);
  gFontNumber = fontNum;
  gFontSize = 10;

  if(!(docStrucHdl = (docStructureHdl) NewHandle(sizeof(docStructure))))
    ExitToShell();
  SetWRefCon(gDocumentWindowRef,(SInt32) docStrucHdl);

  (*docStrucHdl)->printSession  = NULL;
  (*docStrucHdl)->pageFormat    = kPMNoPageFormat;
  (*docStrucHdl)->printSettings = kPMNoPrintSettings;

  // ..... open printing information window

  if(!(gPrintInfoWindowRef = GetNewCWindow(rInfoWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gPrintInfoWindowRef);
  TextFont(fontNum);
  TextSize(10);
  RGBForeColor(&whiteColour);
  RGBBackColor(&blueColour);
  GetWindowPortBounds(gPrintInfoWindowRef,&portRect);
  EraseRect(&portRect);
```

```
  // ..... load and display simulated document

  doGetDocument();

  // ..... event loop

  gDone = false;

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL);
    if(gotEvent)
      doEvents(&eventStructure);
  }
}

// ********************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(128);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ********************************************************************** doQuitAppEvent

 OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************** doGetDocument

void  doGetDocument(void)
{
  docStructureHdl docStrucHdl;
  Rect            portRect, destRect, viewRect;
  Handle          textHdl;

  SetPortWindowPort(gDocumentWindowRef);

  docStrucHdl = (docStructureHdl) GetWRefCon(gDocumentWindowRef);

  GetWindowPortBounds(gDocumentWindowRef,&portRect);
```

```
  destRect = portRect;
  InsetRect(&destRect,4,4);
  destRect.bottom +=4;
  viewRect = destRect;
  (*docStrucHdl)->editTextStrucHdl = TENew(&destRect,&viewRect);

  textHdl = GetResource('TEXT',rText);
  if(textHdl == NULL)
    ExitToShell();

  HLock(textHdl);
  TEInsert(*textHdl,GetHandleSize(textHdl),(*docStrucHdl)->editTextStrucHdl);
  HUnlock(textHdl);

  ReleaseResource(textHdl);

  (*docStrucHdl)->pictureHdl = GetPicture(rPicture);
  if((*docStrucHdl)->pictureHdl == NULL)
    ExitToShell();

  InvalWindowRect(gDocumentWindowRef,&portRect);
}

// ********************************************************************************* doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;

  windowRef = (WindowRef) eventStrucPtr->message;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);
      switch(partCode)
      {
        case inMenuBar:
          doMenuChoice(MenuSelect(eventStrucPtr->where));
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          break;

        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          break;
      }
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoice(MenuEvent(eventStrucPtr));
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;
  }
}

// ********************************************************************************* doUpdate
```

```
void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  GrafPtr   oldPort;
  Rect      portRect;

  windowRef = (WindowRef) eventStrucPtr->message;

  GetPort(&oldPort);

  BeginUpdate(windowRef);

  if(windowRef == gDocumentWindowRef)
    doUpdateDocumentWindow(windowRef);
  else if(windowRef == gPrintInfoWindowRef)
  {
    SetPortWindowPort(gPrintInfoWindowRef);
    GetWindowPortBounds(gPrintInfoWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawPrintInfo();
  }

  EndUpdate(windowRef);

  SetPort(oldPort);
}

// ********************************************************************* doUpdateDocumentWindow

void  doUpdateDocumentWindow(WindowRef windowRef)
{
  Rect            portRect, pictureRect, savedDestRect;
  docStructureHdl docStrucHdl;
  SInt16          savedFontNum, savedFontSize, savedLineHeight, fontNum;

  SetPortWindowPort(windowRef);
  GetWindowPortBounds(gDocumentWindowRef,&portRect);
  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);

  savedDestRect = (*(*docStrucHdl)->editTextStrucHdl)->destRect;
  savedFontNum = (*(*docStrucHdl)->editTextStrucHdl)->txFont;
  savedFontSize = (*(*docStrucHdl)->editTextStrucHdl)->txSize;
  savedLineHeight = (*(*docStrucHdl)->editTextStrucHdl)->lineHeight;

  (*(*docStrucHdl)->editTextStrucHdl)->destRect = portRect;
  InsetRect(&(*(*docStrucHdl)->editTextStrucHdl)->destRect,4,4);
  (*(*docStrucHdl)->editTextStrucHdl)->destRect.bottom += 4;
  GetFNum("\pGeneva",&fontNum);
  (*(*docStrucHdl)->editTextStrucHdl)->txFont = fontNum;
  (*(*docStrucHdl)->editTextStrucHdl)->txSize = 10;
  (*(*docStrucHdl)->editTextStrucHdl)->lineHeight = 13;
  TECalText((*docStrucHdl)->editTextStrucHdl);
  TEUpdate(&portRect,(*docStrucHdl)->editTextStrucHdl);

  (*(*docStrucHdl)->editTextStrucHdl)->destRect = savedDestRect;
  (*(*docStrucHdl)->editTextStrucHdl)->txFont = savedFontNum;
  (*(*docStrucHdl)->editTextStrucHdl)->txFont = savedFontSize;
  (*(*docStrucHdl)->editTextStrucHdl)->lineHeight = savedLineHeight;

  SetRect(&pictureRect,2,2,180,134);
  DrawPicture((*docStrucHdl)->pictureHdl,&pictureRect);
}

// ************************************************************************** doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
```

```
  MenuItemIndex menuItem;
  OSStatus      osStatus;
  Rect          portRect;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      if(menuItem == iPageSetup)
      {
        osStatus = doPageSetUpDialog(gDocumentWindowRef);
        if(osStatus != kPMNoError)
          doErrorAlert(osStatus);
      }

      if(menuItem == iPrint)
      {
        osStatus = doPrintSettingsDialog(gDocumentWindowRef);
        if(osStatus == kPMNoError)
        {
          osStatus = doPrinting(gDocumentWindowRef);
          if(osStatus != kPMNoError)
            doErrorAlert(osStatus);
        }
        else if(osStatus != kPMCancel)
          doErrorAlert(osStatus);
      }

      GetWindowPortBounds(gPrintInfoWindowRef,&portRect);
      InvalWindowRect(gPrintInfoWindowRef,&portRect);

      if(menuItem == iQuit)
        gDone = true;

      break;
  }

  HiliteMenu(0);
}

// ************************************************************* doCreateOrValidatePageFormat

OSStatus  doCreateOrValidatePageFormat (WindowRef windowRef)
{
  docStructureHdl docStrucHdl;
  OSStatus        osStatus     = kPMNoError;
  PMPrintSession  printSession = NULL;
  PMPageFormat    pageFormat   = kPMNoPageFormat;

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);

  // ……..…… create printing session

  osStatus = PMCreateSession(&printSession);

  // ……..…… if necessary, create and store page format object, otherwise validate existing

  if(osStatus == noErr)
```

```
  {
    if((*docStrucHdl)->pageFormat == kPMNoPageFormat)
    {
      osStatus = PMCreatePageFormat(&pageFormat);

      if((osStatus == kPMNoError) && (pageFormat != kPMNoPageFormat))
      {
        osStatus = PMSessionDefaultPageFormat(printSession,pageFormat);

        if(osStatus == kPMNoError)
          (*docStrucHdl)->pageFormat = pageFormat;
      }
      else
      {
        if(osStatus == kPMNoError)
          osStatus = kPMGeneralError;
      }
    }
    else
    {
      osStatus = PMSessionValidatePageFormat(printSession,(*docStrucHdl)->pageFormat,
                                             kPMDontWantBoolean);
    }
  }

  // ..... store printing session, or clean up if error

  if(osStatus == kPMNoError)
    (*docStrucHdl)->printSession = printSession;
  else
  {
    if(pageFormat != kPMNoPageFormat)
      PMRelease(pageFormat);
    if(printSession != NULL)
      PMRelease(printSession);
  }

  HUnlock((Handle) docStrucHdl);

  return osStatus;
}

// ********************************************************************** doPageSetUpDialog

OSStatus  doPageSetUpDialog(WindowRef windowRef)
{
  OSStatus        osStatus = kPMNoError;
  docStructureHdl docStrucHdl;
  Boolean         userClickedOKButton;

  osStatus = doCreateOrValidatePageFormat (windowRef);
  if(osStatus != kPMNoError)
    return osStatus;

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);

  osStatus = PMSessionPageSetupDialog((*docStrucHdl)->printSession,(*docStrucHdl)->pageFormat,
                                      &userClickedOKButton);

  if((*docStrucHdl)->printSession != NULL)
  {
    PMRelease((*docStrucHdl)->printSession);
    (*docStrucHdl)->printSession = NULL;
  }

  HUnlock((Handle) docStrucHdl);
  gDrawPrintSettingsStuff = false;
```

```
  return osStatus;
}

// ********************************************************************** doPrintSettingsDialog

OSStatus  doPrintSettingsDialog(WindowRef windowRef)
{
  OSStatus             osStatus      = kPMNoError;
  docStructureHdl      docStrucHdl;
  PMPrintSettings      printSettings = kPMNoPrintSettings;
  CFStringRef          stringRef;
  PMPrintDialogInitUPP initialisationFunctionUPP;
  Boolean              userClickedPrintButton;

  osStatus = doCreateOrValidatePageFormat (windowRef);
  if(osStatus != kPMNoError)
    return osStatus;

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);

  osStatus = PMCreatePrintSettings(&printSettings);
  if((osStatus == kPMNoError) && (printSettings != kPMNoPrintSettings))
  {
    osStatus = CopyWindowTitleAsCFString(windowRef,&stringRef);
    if(osStatus == noErr)
    {
      osStatus = PMSetJobNameCFString(printSettings,stringRef);
      CFRelease(stringRef);
    }

    if(osStatus == noErr)
      osStatus = PMSessionDefaultPrintSettings((*docStrucHdl)->printSession,printSettings);

    if(osStatus == kPMNoError)
    {
      (*docStrucHdl)->printSettings = printSettings;
      printSettings = kPMNoPrintSettings;
    }
  }

  if(osStatus == kPMNoError)
  {
    if(gRunningOnX)
    {
      PMSetPageRange((*docStrucHdl)->printSettings,1,kPMPrintAllPages);
      PMSetFirstPage((*docStrucHdl)->printSettings,1,false);
      PMSetLastPage((*docStrucHdl)->printSettings,9999,false);
    }
  }

  if(osStatus == kPMNoError)
  {
    initialisationFunctionUPP   = NewPMPrintDialogInitUPP((PMPrintDialogInitProcPtr)
                                                          initialisationFunction);
    gNewItemEvaluateFunctionUPP = NewPMItemUPP((PMItemProcPtr) itemEvaluationFunction);
    gEventFilterUPP             = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

    osStatus = PMSessionPrintDialogInit((*docStrucHdl)->printSession,
                                        (*docStrucHdl)->printSettings,
                                        (*docStrucHdl)->pageFormat,&gPMDialog);
    if(osStatus == kPMNoError)
      osStatus = PMSessionPrintDialogMain((*docStrucHdl)->printSession,
                                          (*docStrucHdl)->printSettings,
                                          (*docStrucHdl)->pageFormat,
                                          &userClickedPrintButton,initialisationFunctionUPP);

    if(osStatus == kPMNoError && !userClickedPrintButton)
      osStatus = kPMCancel;
```

```
    DisposePMPrintDialogInitUPP(initialisationFunctionUPP);
    DisposePMItemUPP(gNewItemEvaluateFunctionUPP);
    DisposeModalFilterUPP(gEventFilterUPP);
  }

  if(osStatus != kPMNoError || osStatus == kPMCancel)
  {
    if((*docStrucHdl)->printSettings != kPMNoPrintSettings)
    {
      PMRelease((*docStrucHdl)->printSettings);
      (*docStrucHdl)->printSettings = kPMNoPrintSettings;
    }
    if((*docStrucHdl)->printSession != NULL)
    {
      PMRelease((*docStrucHdl)->printSession);
      (*docStrucHdl)->printSession = NULL;
    }
  }

  HUnlock((Handle) docStrucHdl);
  gDrawPrintSettingsStuff = userClickedPrintButton;

  return osStatus;
}

// ********************************************************************************** doPrinting

OSStatus  doPrinting(WindowRef windowRef)
{
  docStructureHdl docStrucHdl;
  OSStatus        osStatus = kPMNoError;
  UInt32          firstPage, lastPage, numberOfPages;
  PMRect          adjustedPageRect;
  Rect            pageRect;
  SInt16          page;
  GrafPtr         oldPort, currentPort, printingPort;

  GetPort(&oldPort);
  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);

  // ..... get first and last page to print as set by user in Print dialog

  osStatus = PMGetFirstPage((*docStrucHdl)->printSettings,&firstPage);

  if(osStatus == kPMNoError)
    osStatus = PMGetLastPage((*docStrucHdl)->printSettings,&lastPage);

  // for demo purposes, store first, last page and copies for Some Printing Information window

  gFirstPage = firstPage;
  gLastPage = lastPage;
  PMGetCopies((*docStrucHdl)->printSettings,&gCopies);

  // ..... get actual number of pages in document and, if necessary, adjust last page

  if(osStatus == kPMNoError)
    osStatus = PMGetAdjustedPageRect((*docStrucHdl)->pageFormat,&adjustedPageRect);

  if(osStatus == kPMNoError)
  {
    pageRect.top    = adjustedPageRect.top;
    pageRect.left   = adjustedPageRect.left;
    pageRect.bottom = adjustedPageRect.bottom;
    pageRect.right  = adjustedPageRect.right;

    numberOfPages = doCalcNumberOfPages(windowRef,pageRect);
```

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

// ..... for Mac OS X, set first and last page for progress dialog

if(gRunningOnX)
{
  if(osStatus == kPMNoError)
    osStatus = PMSetFirstPage((*docStrucHdl)->printSettings,firstPage,false);

  if(osStatus == kPMNoError)
    osStatus = PMSetLastPage((*docStrucHdl)->printSettings,lastPage,false);
}

// ..... printing loop

if(osStatus == kPMNoError)
{
  osStatus = PMSessionBeginDocument((*docStrucHdl)->printSession,
                                    (*docStrucHdl)->printSettings,
                                    (*docStrucHdl)->pageFormat);
  if(osStatus == kPMNoError)
  {
    page = 1;

    while((page <= lastPage) && (osStatus == kPMNoError) &&
          (PMSessionError((*docStrucHdl)->printSession) == kPMNoError))
    {
      osStatus = PMSessionBeginPage((*docStrucHdl)->printSession,
                                    (*docStrucHdl)->pageFormat,NULL);
      if(osStatus != kPMNoError)
        break;

      GetPort(&currentPort);

      osStatus = PMSessionGetGraphicsContext((*docStrucHdl)->printSession,
                                             kPMGraphicsContextQuickdraw,
                                             (void **) &printingPort);
      if(osStatus == kPMNoError)
      {
        SetPort(printingPort);
        doDrawPage(windowRef,pageRect,page,numberOfPages);
        SetPort(currentPort);
      }

      osStatus = PMSessionEndPage((*docStrucHdl)->printSession);
      if(osStatus != kPMNoError)
        break;

      page++;
    }
  }

  PMSessionEndDocument((*docStrucHdl)->printSession);
}

// ..... clean up

if((*docStrucHdl)->printSettings != kPMNoPrintSettings)
{
  PMRelease((*docStrucHdl)->printSettings);
  (*docStrucHdl)->printSettings = kPMNoPrintSettings;
}
if((*docStrucHdl)->printSession != NULL)
{
  PMRelease((*docStrucHdl)->printSession);
  (*docStrucHdl)->printSession = NULL;
}
```

```
  HUnlock((Handle) docStrucHdl);
  SetPort(oldPort);

  return osStatus;
}

// ********************************************************************** doCalcNumberOfPages

SInt16  doCalcNumberOfPages(WindowRef windowRef,Rect pageRect)
{
  docStructureHdl docStrucHdl;
  FontInfo        fontInfo;
  Rect            destRect;
  SInt16          heightDestRect, linesPerPage, numberOfPages;

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);

  TextFont(gFontNumber);
  TextSize(gFontSize);
  GetFontInfo(&fontInfo);

  (*(*docStrucHdl)->editTextStrucHdl)->txFont = gFontNumber;
  (*(*docStrucHdl)->editTextStrucHdl)->txSize = gFontSize;
  (*(*docStrucHdl)->editTextStrucHdl)->lineHeight = fontInfo.ascent + fontInfo.descent
                                                  + fontInfo.leading;

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

  (*(*docStrucHdl)->editTextStrucHdl)->destRect = destRect;

  TECalText((*docStrucHdl)->editTextStrucHdl);

  heightDestRect = destRect.bottom - destRect.top;
  linesPerPage = heightDestRect / (*(*docStrucHdl)->editTextStrucHdl)->lineHeight;
  numberOfPages = ((*(*docStrucHdl)->editTextStrucHdl)->nLines / linesPerPage) + 1;

  return(numberOfPages);
}

// ********************************************************************************** doDrawPage

void  doDrawPage(WindowRef windowRef,Rect pageRect,SInt16 pageNumber,SInt16 numberOfpages)
{
  docStructureHdl docStrucHdl;
  TEHandle        docEditTextStrucHdl, pageEditTextStrucHdl;
  PicHandle       pictureHdl;
  Rect            destRect, pictureRect;
  SInt16          heightDestRect, linesPerPage, numberOfLines;
  Handle          textHdl;
  SInt32          startOffset, endOffset;
  Str255          theString;

  TextFont(gFontNumber);
  TextSize(gFontSize);

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  docEditTextStrucHdl  = (*docStrucHdl)->editTextStrucHdl;
  pictureHdl = (*docStrucHdl)->pictureHdl;

  destRect = (*docEditTextStrucHdl)->destRect;
  heightDestRect = destRect.bottom - destRect.top;
  linesPerPage = heightDestRect / (*docEditTextStrucHdl)->lineHeight;
  numberOfLines = (*docEditTextStrucHdl)->nLines;

  startOffset = (*docEditTextStrucHdl)->lineStarts[(pageNumber - 1) * linesPerPage];
  if(pageNumber == numberOfpages)
    endOffset = (*docEditTextStrucHdl)->lineStarts[numberOfLines];
```

```
  else
    endOffset = (*docEditTextStrucHdl)->lineStarts[pageNumber * linesPerPage];

  pageEditTextStrucHdl = TENew(&destRect,&destRect);
  textHdl = (*docEditTextStrucHdl)->hText;
  HLock(textHdl);
  TEInsert(*textHdl + startOffset,endOffset - startOffset,pageEditTextStrucHdl);
  TEDispose(pageEditTextStrucHdl);

  if(pageNumber == 1)
  {
    SetRect(&pictureRect,destRect.left,destRect.top,
            destRect.left + ((*pictureHdl)->picFrame.right - (*pictureHdl)->picFrame.left),
            destRect.top + ((*pictureHdl)->picFrame.bottom - (*pictureHdl)->picFrame.top));
    DrawPicture(pictureHdl,&pictureRect);
  }

  MoveTo(destRect.left,pageRect.bottom - 25);
  NumToString((SInt32) pageNumber,theString);
  DrawString(theString);
}

// ************************************************************************** doDrawPrintInfo

void  doDrawPrintInfo(void)
{
  docStructureHdl docStrucHdl;
  OSStatus        osStatus = kPMNoError;
  PMRect          theRect;
  Str255          s2, s3;
  PMResolution    resolution;
  double          scale;
  PMOrientation   orientation;

  docStrucHdl = (docStructureHdl) GetWRefCon(gDocumentWindowRef);
  if((*docStrucHdl)->pageFormat == kPMNoPageFormat)
    return;

  HLock((Handle) docStrucHdl);

  MoveTo(20,25);
  TextFace(bold);
  DrawString("\pFrom PMPageFormat Object:");
  TextFace(normal);

  PMGetAdjustedPaperRect((*docStrucHdl)->pageFormat,&theRect);
  NumToString((SInt32) theRect.top,s2);
  NumToString((SInt32) theRect.left,s3);
  doDrawRectStrings("\pPaper Rectangle (top,left):",20,45,s2,190,45,s3);
  NumToString((SInt32) theRect.bottom,s2);
  NumToString((SInt32) theRect.right,s3);
  doDrawRectStrings("\pPaper Rectangle (bottom,right):",20,60,s2,190,60,s3);

  PMGetAdjustedPageRect((*docStrucHdl)->pageFormat,&theRect);
  NumToString((SInt32) theRect.top,s2);
  NumToString((SInt32) theRect.left,s3);
  doDrawRectStrings("\pPage Rectangle (top,left):",20,75,s2,190,75,s3);
  NumToString((SInt32) theRect.bottom,s2);
  NumToString((SInt32) theRect.right,s3);
  doDrawRectStrings("\pPage Rectangle (bottom,right):",20,90,s2,190,90,s3);

  PMGetResolution((*docStrucHdl)->pageFormat,&resolution);
  MoveTo(20,105);
  DrawString("\pDrawing Resolution (Vertical):");
  NumToString((SInt32) resolution.vRes,s2);
  MoveTo(190,105);
  DrawString(s2);
  DrawString("\p dpi");
  MoveTo(20,120);
```

```
  DrawString("\pDrawing Resolution (Horizontal):");
  NumToString((SInt32) resolution.hRes,s2);
  MoveTo(190,120);
  DrawString(s2);
  DrawString("\p dpi");

  PMGetScale((*docStrucHdl)->pageFormat,&scale);
  MoveTo(20,135);
  DrawString("\pScale:");
  NumToString((SInt32) scale,s2);
  MoveTo(190,135);
  DrawString(s2);
  DrawString("\p%");

  PMGetOrientation((*docStrucHdl)->pageFormat,&orientation);
  MoveTo(20,150);
  DrawString("\pPage Orientation:");
  MoveTo(190,150);
  if(orientation == kPMPortrait)
    DrawString("\pPortrait");
  else if(orientation == kPMLandscape)
    DrawString("\pLandscape");

  if(gDrawPrintSettingsStuff)
  {
    MoveTo(20,170);
    TextFace(bold);
    DrawString("\pFrom PMPrintSettings Object:");
    TextFace(normal);

    MoveTo(20,190);
    DrawString("\pFirst Page:");
    NumToString((SInt32) gFirstPage,s2);
    MoveTo(190,190);
    DrawString(s2);

    MoveTo(20,205);
    DrawString("\pLast Page:");
    NumToString((SInt32) gLastPage,s2);
    MoveTo(190,205);
    DrawString(s2);

    MoveTo(20,220);
    DrawString("\pNumber of Copies:");
    NumToString((SInt32) gCopies,s2);
    MoveTo(190,220);
    DrawString(s2);
  }

  HUnlock((Handle) docStrucHdl);
}

// ********************************************************************** doDrawRectStrings

void  doDrawRectStrings(Str255 s1,SInt16 x1,SInt16 y1,Str255 s2,SInt16 x2,SInt16 y2,Str255 s3)
{
  MoveTo(x1,y1);
  DrawString(s1);
  MoveTo(x2,y2);
  DrawString("\p(");
  DrawString(s2);
  DrawString("\p,");
  DrawString(s3);
  DrawString("\p)");
}

// ************************************************************************** doErrorAlert

void  doErrorAlert(OSStatus errorType)
```

```
{
  Str255 theString, errorString = "\pCarbon Printing Manager Error ";
  SInt16 itemHit;

  NumToString((SInt32) errorType,theString);
  doConcatPStrings(errorString,theString);

  StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
}

// ********************************************************************** doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16  appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

// ****************************************************************************************
// PrintDialogAppend.c
// ****************************************************************************************

// ..... includes

#include "CarbonPrinting.h"

// ..... global variables

SInt32    gFirstAppendedItemNo;
PMItemUPP gOldItemEvaluateFunctionUPP;

extern PMDialog       gPMDialog;
extern PMItemUPP      gNewItemEvaluateFunctionUPP;
extern ModalFilterUPP gEventFilterUPP;
extern SInt16         gFontNumber;
extern SInt16         gFontSize;

// ********************************************************************** initialisationFunction

void  initialisationFunction(PMPrintSettings printSettings,PMDialog *pmDialog)
{
  OSStatus   osStatus = kPMNoError;
  DialogRef  dialogRef;
  Handle     ditlHdl;
  SInt16     numberOfExistingItems, numberOfMenuItems;
  MenuRef    menuRef;
  ControlRef controlRef;
  Str255     itemName;

  *pmDialog = gPMDialog;

  osStatus = PMGetDialogPtr(*pmDialog,&dialogRef);

  if(osStatus == kPMNoError)
  {
    // ..... append DITL

    ditlHdl = GetResource('DITL',rPrintDialogAppendDITL);
    numberOfExistingItems = CountDITL(dialogRef);
    AppendDITL(dialogRef,ditlHdl,appendDITLBottom);
    gFirstAppendedItemNo = numberOfExistingItems + 1;
```

```
    // ..... create font menu and attach to popup button, set current font to first item

    menuRef = NewMenu(mFont,NULL);
    CreateStandardFontMenu(menuRef,0,0,0,NULL);
    GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo,&controlRef);
    SetControlMinimum(controlRef,1);
    numberOfMenuItems = CountMenuItems(menuRef);
    SetControlMaximum(controlRef,numberOfMenuItems);
    SetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuRefTag,
                   sizeof(menuRef),&menuRef);
    GetMenuItemText(menuRef,1,itemName);
    GetFNum(itemName,&gFontNumber);

    // ..... set second radio button to on state and set current font size

    GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 2,&controlRef);
    SetControlValue(controlRef,1);
    gFontSize = 12;

    // ..... switch fractional widths off

    GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 4,&controlRef);
    SetControlValue(controlRef,0);
    SetFractEnable(false);
  }

  if(osStatus == kPMNoError)
    osStatus = PMGetItemProc(*pmDialog,&gOldItemEvaluateFunctionUPP);
  if(osStatus == kPMNoError)
    osStatus = PMSetItemProc(*pmDialog,gNewItemEvaluateFunctionUPP);

  if(osStatus == kPMNoError)
    PMSetModalFilterProc(*pmDialog,gEventFilterUPP);

  if(osStatus != kPMNoError)
    doErrorAlert(osStatus);
}

// ********************************************************************* itemEvaluationFunction

void  itemEvaluationFunction(DialogRef dialogRef,SInt16 itemHit)
{
  SInt16     localizedItemNo, controlValue;
  ControlRef controlRef;
  MenuRef    menuRef;
  Str255     itemName;

  localizedItemNo = itemHit - gFirstAppendedItemNo + 1;

  if(localizedItemNo > 0)
  {
    if(localizedItemNo == iPopupButton)
    {
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo,&controlRef);
      controlValue = GetControlValue(controlRef);
      GetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuHandleTag,
                     sizeof(menuRef),(Ptr) &menuRef,NULL);
      GetMenuItemText(menuRef,controlValue,itemName);
      GetFNum(itemName,&gFontNumber);
    }
    else if(localizedItemNo >= iRadioButton10pt && localizedItemNo <= iRadioButton14pt)
    {
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 1,&controlRef);
      SetControlValue(controlRef,0);
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 2,&controlRef);
      SetControlValue(controlRef,0);
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 3,&controlRef);
      SetControlValue(controlRef,0);
```

```
      GetDialogItemAsControl(dialogRef,itemHit,&controlRef);
      SetControlValue(controlRef,1);

      if(localizedItemNo == iRadioButton10pt)
        gFontSize = 10;
      else if(localizedItemNo == iRadioButton12pt)
        gFontSize = 12;
      else if(localizedItemNo == iRadioButton14pt)
        gFontSize = 14;
    }
    else if(localizedItemNo == iCheckboxFracWidths)
    {
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 4,&controlRef);
      SetControlValue(controlRef,!GetControlValue(controlRef));
      SetFractEnable(GetControlValue(controlRef));
    }
  }
  else
  {
    InvokePMItemUPP(dialogRef,itemHit,gOldItemEvaluateFunctionUPP);
  }
}

// ********************************************************************************** eventFilter

Boolean  eventFilter(DialogRef dialogRef,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean handledEvent;
  GrafPtr oldPort;

  handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != GetDialogWindow(dialogRef)))
  {
    doUpdate(eventStrucPtr);
  }
  else
  {
    GetPort(&oldPort);
    SetPortDialogPort(dialogRef);

    handledEvent = StdFilterProc(dialogRef,eventStrucPtr,itemHit);

    SetPort(oldPort);
  }

  return handledEvent;
}

// ********************************************************************************************
```

Demonstration Program CarbonPrinting Comments

When the program is run, the user should:

- Choose Page Setup… from the File menu, make changes in the Page Setup dialog, and observe the information, extracted from the PMPageFormat object, drawn in the window titled Some Printing Information.

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

The user should print the document several times using different page size, scaling, and orientation settings in the Page Setup dialog (noting the changed information in the Printing Settings window), and occasionally limiting the number of changes printed by changing the start-page and end-page settings in the Print dialog.

The window in which the "document" is displayed is titled Simulated Document because, in this demonstration, the document is not loaded from a file. Rather, a document is simulated using text from a 'TEXT' resource and a picture from a 'PICT' resource. This approach is used in order to keep that part of the source code not related to printing per se to a minimum. For the same reason, the (flattened) PMPageFormat object is not saved to the resource fork of a document file in this demonstration. (The demonstration program at Chapter 19 shows how to do this.)

Printing.h

defines

Constants are established for the resource IDs of a 'PICT' and 'TEXT' resources used for the printing demonstration, and for the 'DITL' resource containing the items to be appended to the Print dialog. Five constants are established for the item numbers of the items in the 'DITL' resource. kMargin is used to set the margins for the printed page.

typedefs

A handle to a structure of type docStructure will be stored in the Window object for the Simulated Document window. The fields of this structure will be assigned PMPrintSession, PMPageFormat, and PMPrintSettings objects, together with handles to the 'TEXT' and 'PICT' resources referred to above.

CarbonPrinting.c

Global Variables

gFontNumber will be assigned the current font number. gFontSize will be assigned the current font size. Both of these values can be changed by the user via the items added to the Print dialog.

gNewItemEvaluateFunctionUPP and gEventFilterUPP will be assigned a universal procedure pointer to, respectively, the item evaluation function and event filter function for the customised Print dialog. gPMDialog will be assigned a pointer to a PMDialog object for the customised Print dialog.

The last four variables are incidental to the demonstration, and will be used to store information to be displayed in the second window (titled Some Printing Information) created by the program.

main

After the Simulated Document window is created, its font is set to Geneva 10 point and the global variables gFontNumber and gFontSize are set to reflect this. A document structure is then created and associated with the window. The three fields of the document structure which will be assigned Carbon Printing Manager objects are then initialised.

The Some Printing Information window is then created.

The call to doGetDocument displays the simulated document in the Simulated Document window.

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

doGetDocument

doGetDocument creates a monostyled TextEdit structure, assigns the handle to this structure to the relevant field of the Simulated Document window's document structure, loads a 'TEXT' resource, and inserts

it into a TextEdit structure. (The act of insertion causes the text to be drawn on the screen.) A 'PICT' resource is then loaded and its handle assigned to the relevant field of the Simulated Document window's document structure. The call to InvalWindowRect invalidates the content region, generating an update event which, as will be seen, causes the picture to be drawn in the window.

TextEdit is not addressed until Chapter 21; however, to facilitate an understanding of the TextEdit aspects of this program, it is sufficient at this stage to understand that a monostyled TextEdit structure contains, amongst others, 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 actual text.
txFont	The font number of the font to be used,
txSize	The size of the font in points.
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.

Note that the destination and view rectangles were passed in the call to TENew, these rectangles having been defined in the preceding five lines of code.

doUpdateDocumentWindow

As will be seen, two of the functions directly related to printing operations will change the values in the TextEdit structure's destRect, txFont, txSize, and lineHeight fields. The middle block of the doUpdateDocumentWindow code simply resets the values in these fields back to those that existed after doGetDocument was called, and then calls TECalText to re-calculate the line starts and TEUpdate to draw the text.

The second blocks saves the values in the TextEdit structure's destRect, txFont, txSize, and lineHeight fields as they existed before the update event was received and the fourth block restores these settings. The last two lines redraw the picture in the window.

doMenuChoice

If the user chose Page Setup… from the File menu, doPageSetupDialog is called to display the Page Setup dialog, handle user interaction within the dialog, and record the settings made by the user. If an error is returned, doErrorAlert is called to present an alert advising of the error number.

If the user chose Print… from the File menu, doPrintSettingsDialog is to display the Print dialog, handle user interaction within the dialog, and record the settings made by the user. If no error is returned, doPrinting is then called to perform the printing. If doPrinting returns an error, doErrorAlert is called to present an alert advising of the error number.

If doPrintSettingsDialog returns an error, doPrinting is not called and doErrorAlert is called to present an alert advising of the error number.

doCreateOrValidatePageFormat

doCreateOrValidatePageFormat is the first of the major printing functions. It is called, as their first action, by doPageSetUpDialog and doPrintSettingsDialog – that is, whenever the user chooses Page Setup… or Print… from the File menu. The main purpose of this function is to create a PMPageFormat object, assign default parameter values to that object, and associate the object with the document's window or, if the object has previously been created, simply validate the existing object.

The call to PMCreateSession creates a printing session object.

If a PMPageFormat object is not currently assigned to the pageFormat field of the window's document structure, PMCreatePageFormat and PMSessionDefaultPageFormat are called to create a PMPageFormat object and assign default parameters to it. The object is then associated with the document's window by assigning the object to the relevant field of the window's document structure.

On the other hand, if a PMPageFormat object is currently assigned to the pageFormat field of the window's document structure, PMSessionValidatePageFormat is called to validate the object within the context of the current printing session.

If no errors occurred, the PMPrintSession object is assigned to the relevant field of the document window's document structure, otherwise both the PMPrintSession and PMPageFormat objects are released.

doPageSetUpDialog

doPageSetUpDialog is called when the user chooses Page Setup… from the File menu. Its purpose is to present the Page Setup dialog and modify the page setup information stored in the PMPageFormat object associated with the document's window according to settings made by the user within the dialog.

The call to doCreateOrValidatePageFormat ensures that PMPageFormat and PMPrintSession objects have been created and assigned to the relevant field of the window's document structure. The call to PMSessionPageSetupDialog then presents the Page Setup dialog and handles all user interaction within the dialog. If the user clicks the OK push button to dismiss the dialog, the PMPageFormat object will be updated to reflect any changed settings made by the user. No updating occurs if the user clicks the Cancel push button to dismiss the dialog.

When the dialog has been dismissed, PMRelease is called to release the PMPrintSession object.

doPrintSettingsDialog

doPrintSettingsDialog is called when the user chooses Print… from the File menu. Its main purpose is to create a PMPrintSettings object and associate it with the document's window, and present the Print dialog and modify the print settings information stored in the PMPrintSettings object according to changed settings made by the user within the dialog.

The call to doCreateOrValidatePageFormat ensures that PMPageFormat and PMPrintSession objects have been created and assigned to the relevant field of the window's document structure. The call to PMCreatePrintSettings then creates a PMPrintSettings object.

The call to PMSetJobNameCFString sets the print job name used by the printing system. PMSessionDefaultPrintSettings is then called to assign default parameter values (number of copies, page range, etc.) to the PMPrintSettings object, following which the PMPrintSettings object is assigned to the relevant field of the document window's document structure.

The next block executes only if the program is running on Mac OS X. PMSetPageRange sets the valid range of pages that can be printed to 1-32000. (If the user enters values outside this range in the Print dialog, the Carbon Printing Manager displays an "out-of-range" alert.) The next two calls set the default first and last page numbers to be drawn in the relevant edit text fields in the Print dialog. Note that, if kPMPrintAllPages is passed in PMSetLastPage's last parameter, the All radio button in the Print dialog will be selected when the dialog is displayed.

The Print dialog to be presented is to be customised. Accordingly, the first three lines inside the next if block create universal procedure pointers for the initialisation, item evaluation, and event filter functions contained in the source code file PrintDialogAppend.c.

PMSessionPrintDialogInit is called to initialise a custom Print dialog. On return, the global variable gPMDialog contains a pointer to an initialised PMDialog object, ready for customisation. This pointer is assigned to a global variable because the PMSessionPrintDialogMain function does not include a parameter for passing this PMDialog object to the dialog initialization function.

PMSessionPrintDialogMain presents the customised Print dialog, the universal procedure pointer to the initialisation function being passed in the fifth parameter.

If the user clicked the Print push button, when PMSessionPrintDialogMain returns, the PMPrintSettings object will contain information on the settings displayed in the Print dialog (except for the settings in the customised section of the dialog).

If an error occurred, or if the the user clicked the Cancel push button in the Print dialog, the PMPrintSettings and PMPrintSession objects are released and disassociated from the window. In addition, within the function doMenuChoice, the call to the function doPrinting will not occur.

doPrinting

doPrinting contains the printing loop.

After the current graphics port is saved for later restoration, PMGetFirstPage and PMGetLastPage are called to get the first and last page to print, as set in the Print dialog.

The next block is incidental to the demonstration, simply storing the first page, last page, and number of copies information retrieved from the PMPrintSettings object in global variables so that they can later be drawn in the Some Printing Information window.

PMGetAdjustedPageRect is then called to get the page rectangle, taking account of page orientation setting, scale setting, and drawing resolution. The double values in the fields of this PMRect structure are assigned to the (SInt16) fields of a normal Rect structure before that rectangle is passed to a function which calculates the actual number of pages in the document.

If the calculated actual number of pages is less than the value returned by the call to PMGetLastPage, the value in the variable lastPage is changed to the calculated number of pages.

For Mac OS X only, PMSetFirstPage and PMSetLastPage are called to set the first and actual last page in the PMPrintSettings object. This ensures that the page number information displayed in the progress dialog invoked during printing will be correct.

The printing loop is now entered. The first action is to call PMSessionBeginDocument to establish a graphics context (PMPrintContext object) for imaging the document. As will be seen, a function exists to obtain the graphics port (that is, the printing port) associated with this object.

The while loop spools all of the pages in the document, relying on the Carbon Printing Manager to print the correct page range. Within the loop:

- GetPort is called to save the current graphics port.
- PMSessionBeginPage is called to initialise the printing port. (Note that, for Mac OS 8/9, this function may also be used to initialise a scaling rectangle for drawing the page, in which case the address of a PMRect passed in the last parameter will be passed to the printer driver.)
- PMSessionGetGraphicsContext is called to obtain the current printing port, and SetPort is called to set this port as the current port.
- doDrawPage is called to draw the specified page in the current printing port.
- SetPort is called to restore the saved graphics port.
- PMSessionEndPage is called to print the page.

After the pages have been spooled, PMSessionEndDocument is called to close the printing port. On Mac OS X, this call also dismisses the progress dialog.

Note that the printing loop does not have to concern itself with the number of copies, since this is handled automatically by the Carbon Printing Manager.

Finally, the PMPrintSettings and PMPrintSession objects are released and disassociated from the document's window, and the graphics port saved at function entry is restored.

doCalcNumberOfPages

doCalcNumberOfPages is called by doPrinting to calculate the actual number of pages in the document based on the page rectangle passed to it.

The first line retrieves the handle to the specified window's document structure. The next two lines set the current font and font size to the font number and size set by the user in the appended items in the Print dialog. This allows the call to GetFontInfo to retrieve some relevant information about the font.

The next four lines change the values in the txFont, txSize, and lineHeight fields of the TextEdit structure whose handle is stored in the window's document structure. (Note that information obtained by the GetFontInfo call is used to calculate line height.)

The next three lines change the rectangle stored in the destRect field of the TextEdit structure to one 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.)

With these changes made, TECalText is called to recalculate the line starts. In addition to changing the values in the lineStarts array in the TextEdit structure, this call will assign the new total number of lines to the nLines field.

The matter of the actual calculation of the number of pages now follows. The first line in the last block gets the height of the previously defined destination 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.

doDrawPage

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

The first action is to set the printing graphics port's font and and font size to the font number and size set by the user in the appended items in the Print dialog.

The next block retrieves a handle to the specified window's document structure, allowing handles to the TextEdit structure and picture to be retrieved.

In the next block, the destination rectangle in the destRect field of the TextEdit structure is assigned to a local variable, the height of this rectangle is assigned to a local variable, the lines per page is calculated and assigned to a local variable, and the total number of lines is assigned to a local variable.

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.

The call to TENew creates a new monostyled TextEdit structure with the previously defined destination rectangle passed in both the destination and view rectangle parameters. The following line gets a handle to the actual text in the TextEdit structure. This handle is then locked preparatory to a call to TEInsert. Using the offsets previously calculated, TEInsert then inserts the text for the current page into the newly created TextEdit structure, an action which causes that text to be drawn in the printing graphics port. The text having been drawn, the TextEdit structure is then disposed of.

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 the original page rectangle.

doDrawPrintInfo and doDrawRectStrings

doDrawPrintInfo is called when an update event is received for the Some Printing Information window. Carbon Printing Manager accessor functions are then used to obtain information from the PMPageFormat object associated with the document's window and draw that information in the window. In addition, information retrieved from the PMPrintSettings, and saved to global variables within the function doPrinting, is drawn in the window if the global variable doDrawPrintSettingsStuff is set to true.

doErrorAlert

doErrorAlert is called when the printing functions return an error other than the "user cancelled" error. An alert showing the error code is displayed.

PrintDialogAppend.c

initialisationFunction

Recall that, in the function doPrintSettingsDialog, a universal procedure pointer to initialisationFunction was passed in the myInitProc parameter of the PMSessionPrintDialogMain call. PMSessionPrintDialogMain thus calls this function.

Recall also that the pointer to the initialised PMDialog object was assigned to a global variable (gPMDialog) because the PMSessionPrintDialogMain function does not include a parameter for passing a PMDialog object to a dialog initialisation function. At the first line, the pointer to the PMDialog object is copied to the initialisation function's formal parameter pmDialog.

The call to PMGetDialogPtr gets a reference to the Print dialog dialog object.

The DITL to be appended to the dialog contains the following items:

- A pop-up menu for font selection.
- Three radio buttons for font size selection.

- A checkbox for selecting fractional widths on or off.

- A primary group box (text title variant).

GetResource loads the specified 'DITL' resource and gets a handle to it. CountDITL counts the current number of items in the Print dialog. AppendDITL then appends the new items to the dialog. For some printers on Mac OS 8/9, this causes the dialog to expand downwards to accommodate the added items. For others (for, example, the LaserWriter 8), and on Mac OS X, the result of the AppendDITL call is that a pane is created for the items and the name of the application is inserted into the menu of a pop-up group box. When the item containing the application's name is chosen from the pop-up menu, the pane is displayed and the appended items are accessible.

The global variable gFirstAppendedItemNo is then assigned the item number in the new Print dialog item list of the first appended item (the pop-up menu button). This will be required by the function itemEvaluationFunction.

The next block calls CreateStandardFontMenu to create the menu for the pop-up menu button, which is then assigned to the pop-up menu button by SetControlData. Note that the font number for the first item in the menu is assigned to the global variable gFontNumber.

The next block selects the second radio (12pt) button and sets the global variable gFontSize to 12. The next block unchecks the checkbox and sets fractional widths to off.

The printer driver's item evaluation function will be called upon by the item evaluation function (itemEvaluationFunction) to handle mouse-downs in the Print dialog's standard items. Accordingly, PMGetItemProc is called to assign the universal procedure pointer to the driver's evaluation function to a global variable for later use. The call to PMSetItemProc makes itemEvaluationFunction the current item evaluation function.

Finally, the call to PMSetModalFilterProc makes the application-defined (callback) function eventFilter the event filter function for the Print dialog.

itemEvaluationFunction

itemEvaluationFunction handles item hits in the Print 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 Print 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 Print 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. The use of a the event filter is optional. Its use in this program simply allows the Print dialog, together with windows belonging to background applications, to receive update events (required only on Mac OS 8/9).

> The Page Setup and Print dialogs in this demonstration program are application-modal. This is because the AppendDITL method used to customise the Print dialog prevents that dialog from being created as a window-modal (sheet) dialog. The basic modifications required to cause the dialogs to be window-modal are as follows:
>
> - Display the Print dialog using the function PMSessionPrintDialog and eliminate all code relating to dialog customisation.

- Immediately before the calls to PMSessionPageSetupDialog and PMSessionPrintDialog, call the function PMSessionUseSheets, passing the parent window's reference in the documentWindow parameter and a universal procedure pointer to an application-defined (callback) function in the

- Add two application-defined (callback) functions which perform the actions required immediately following the dismissal of the dialogs, and modify the existing post-dismissal code accordingly.