I n s i d e   C a r b o n

Adopting the Carbon Printing Manager

Preliminary

5/17/00

Contents

Figures, Listings, and Tables

Concepts

This document provides important information about changes to the Printing Manager interface for Carbon applications. This information is intended to help experienced Macintosh developers convert existing Mac OS 8 and 9 applications into Carbon applications that can print on Mac OS X as well as on Mac OS 8 and 9. You should read this document if you are creating a Carbon application that supports printing.

In order to understand the information presented here, you should be familiar with the Classic Printing Manager interface, as well as QuickDraw and the Dialog Manager. Documentation for these technologies is available on Apple's website at

http://developer.apple.com/techpubs/macos8

Overview

Mac OS X introduces a new printing system that is significantly different from the Classic Printing Manager available on Mac OS 8 and 9. The Carbon Printing Manager provides a bridge between these two architectures, allowing Carbon applications to print on both platforms.

On Mac OS 8 and 9, the Carbon Printing Manager is built into CarbonLib. Carbon applications running on Mac OS 8 and 9 continue to print using Classic printer drivers, and most Carbon Printing Manager functions call through to their Classic counterparts. On Mac OS X, the Printing Framework belongs to the Carbon umbrella framework. When running on Mac OS X, Carbon applications automatically use the new printing system.

The Carbon Printing Manager is designed to make the transition from Classic to Carbon as straightforward as possible. While the function names have changed, there is a high level of correspondence between Carbon and Classic printing functions, and the basic print loop model has been preserved.

Because it does not fundamentally alter the structure of your application's print loop, adopting the Carbon Printing Manager is relatively simple. The most significant change is that the data structures corresponding to the Classic print record are opaque in Carbon—meaning their contents are hidden from applications. There are new functions for accessing fields within these structures. Most other Carbon Printing Manager functions are direct replacements for Classic Printing Manager functions.

Summary of Changes

The Carbon Printing Manager interface for applications is defined in the header files `PMApplication.h`, `PMCore.h`, and `PMDefinitions.h`. The Classic Printing Manager interface, defined in `Printing.h`, is not used by Carbon applications.

The following list summarizes the changes you need to be aware of as you convert your existing printing code to use the Carbon Printing Manager. These topics are discussed in more detail in subsequent sections.

- The Classic print record (`TPrint`) is replaced by two opaque objects: a `PMPrintSettings` object and a `PMPageFormat` object. You create these objects using the `PMCreatePrintSettings` and `PMCreatePageFormat` functions, which return a reference that you can pass to other Carbon Printing Manager functions to obtain information stored in the objects.
- Your application must not make assumptions about the size or content of the `PMPrintSettings` and `PMPageFormat` objects. Applications can attach extended data to both these objects using new Carbon Printing Manager functions, but applications must not assume a specific size when storing or retrieving these objects with documents.
- The Carbon Printing Manager provides functions for flattening and restoring the `PMPrintSettings` and `PMPageFormat` objects. In most cases, your application should store only the `PMPageFormat` object. If older versions of your application

store a Classic print record with a saved document, you may continue to do so for backward compatibility. However, some information may be lost when converting a `PMPageFormat` object into a Classic print record, due to limitations of the Classic print record structure.

- The Classic printing port record (`TPrPort`) is replaced by an opaque `PMPrintContext` object. The Carbon Printing Manager provides functions for obtaining the graphics port associated with a `PMPrintContext` object.
- On Mac OS X, Carbon applications can initiate multiple, simultaneous printing tasks. Each printing task is referred to as a **session**, and each printing session is independent of other printing sessions. On Mac OS 8 and 9, applications are limited to a single printing session.
- The Carbon Printing Manager enforces an order in which some routines can be called. Any routine used out of order will return the result code `kPMOutOfScope`.
- The “style” dialog box is now called the Page Setup dialog box, and the “job” dialog box is now called the Print dialog box.
- If your application requires customizing the Page Setup or Print dialog boxes, you should consider creating a printer dialog extension (PDE). Printer dialog extensions provide more customization capability, and are required to support document-modal dialog boxes in Mac OS X. For more information, see *Extending Printing Dialog Boxes in Mac OS X*, available on the Apple website.
- The `PrPicFile` function and the “deferred” printing style are not supported in Carbon. All print records must use the “draft” style, and printer drivers must perform their own spooling or, on Mac OS 8 and 9, use the Desktop Printer Spooler. See Tech Note 1097 for information about the Desktop Printer Spooler.
- Low-level driver functions such as `PrLoadDriver` are not supported. See “New and Changed Printing Functions” (page 23) for a list of unsupported functions.

Understanding Printing Sessions

On Mac OS X, Carbon applications can initiate multiple, simultaneous printing tasks. Each printing task is referred to as a **session**, and each printing session is independent of other printing sessions. Printing sessions can run in separate threads, or you can create multiple sessions within a single-threaded application.

Printing sessions are also supported on Mac OS 8 and 9, but each application can have only one printing session running in a single thread, due to limitations of the underlying Classic printing architecture.

To support printing sessions, the Carbon Printing Manager API is conceptually divided into three groups: session functions, non-session functions, and universal functions. Session functions allow you to create and manage printing sessions. These functions take a `PMPrintSession` parameter, which is an opaque object that uniquely identifies a particular printing session.

The non-session functions offer an alternative API that provides similar services, but does not support multithreading or simultaneous printing sessions. These functions closely mirror their Classic counterparts, and operate identically on all Carbon platforms, but they also inherit most of the limitations of the Classic Printing Manager. Apple therefore strongly recommends that you adopt the session functions instead.

You must not mix session and non-session functions within your application. The Carbon Printing Manager header files are designed to prevent this by using conditional macros to enable only one or the other. The session functions are enabled by default, but you can enable the non-session functions by adding the following statement to one of your source files before including the Carbon Printing Manager headers:

```
#define PM_USE_SESSION_APIS 0
```

The universal functions, such as page format and print settings accessors, are available whether you are using session or non-session functions.

Understanding Calling Sequence and Scope

The Carbon Printing Manager enforces a sequence of steps in the printing loop, and defines a valid scope for each printing function. This means that your application must call certain functions before calling others. Functions used out of sequence will return an error result of `kPMOutOfScope`.

The rules for calling sequence and scope are different for session and non-session printing functions. The following sections describe these requirements in detail.

Sequence and Scope: Session Functions

Figure 1-1 (page 18) shows the calling sequence and scope requirements you need to be aware of when using session functions. The diagram shows, for example, that you can call `PMSessionEnableColorSync` only after `PMSessionBeginPage`.

In general, functions may be called in any order with respect to other functions at the same or lower scope level (represented in the diagram by indentation). This does not mean that you should call `PMSessionConvertOldPrintRecord` in the middle of your page drawing code, it simply means that the Carbon Printing Manager does not restrict you from doing so. The exception is functions relating to dialog boxes. These can be called at any time after `PMCreateSession`, except after calling `PMSessionBeginDocument`.

The universal functions `PMIsPostScriptDriver` through `PMPrinterGetLanguageInfo` are also included in the diagram, to show their scope restrictions with respect to the session functions.

Figure 1-1 Calling sequence and scope of session functions

```
PMSessionSetIdleProc
PMCreateSession
    PMSessionGetCurrentPrinter
    PMSessionIsDocumentFormatSupported
    PMSessionGetDocumentFormatSupported
    PMSessionGetDocumentFormatGeneration
    PMSessionSetDocumentFormatGeneration
    PMSessionDefaultPageFormat
    PMSessionValidatePageFormat
    PMSessionDefaultPrintSettings
    PMSessionValidatePrintSettings
    PMSessionConvertOldPrintRecord
    PMSessionMakeOldPrintRecord
    PMSessionGeneral
    PMSessionError
    PMSessionSetError

    PMSessionPageSetupDialog
    PMSessionPrintDialog
    PMSessionPageSetupDialogInit
    PMSessionPrintDialogInit
    PMSessionPrintDialogMain          Must be called before PMSessionBeginDocument.
    PMSessionPageSetupDialogMain
    PMSessionPageSetupDialogMain
    PMSessionUseSheets

    PMIsPostScriptDriver
    PMGetLanguageInfo
    PMGetDriverCreator
    PMGetDriverReleaseInfo
    PMGetPrinterResolutionCount
    PMGetPrinterResolution
    PMGetIndexedPrinterResolution
    PMPrinterGetDriverCreator
    PMPrinterGetDriverReleaseInfo
    PMPrinterGetPrinterResolutionCount
    PMPrinterGetPrinterResolution
    PMPrinterGetIndexedPrinterResolution
    PMPrinterGetDescriptionURL
    PMPrinterGetLanguageInfo

    PMSessionBeginDocument
        PMSessionPostScriptBegin
        PMSessionPostScriptEnd
        PMSessionPostScriptHandle
        PMSessionPostScriptData
        PMSessionPostScriptFile

        PMSessionBeginPage
            PMSessionGetGraphicsContext
            PMSessionEnableColorSync
            PMSessionDisableColorSync
        PMSessionEndPage
    PMSessionEndDocument
```

Sequence and Scope: Non-Session Functions

Figure 1-2 (page 20) shows the calling sequence and scope requirements for non-session printing functions. The universal functions `PMIsPostScriptDriver` through `PMPrinterGetLanguageInfo` are included in the diagram to show their scope restrictions when used with non-session functions.

Figure 1-2 Calling sequence and scope of non-session functions

```
PMBegin
    PMDefaultPageFormat
    PMValidatePageFormat
    PMDefaultPrintSettings
    PMValidatePrintSettings
    PMConvertOldPrintRecord
    PMMakeOldPrintRecord
    PMSetIdleProc
    PMGeneral
    PMError
    PMSetError

    PMPageSetupDialog
    PMPrintDialog
    PMPrintDialogInit
    PMPrintDialogInitWithPageFormat    Must be called before PMBeginDocument.
    PMPrintDialogMain
    PMPageSetupDialogInit
    PMPageSetupDialogMain

    PMGetDialogPtr
    PMGetModalFilterProc
    PMSetModalFilterProc
    PMGetItemProc                      A valid PMDialog object is required before
    PMSetItemProc                      calling these functions.
    PMGetDialogAccepted
    PMSetDialogAccepted
    PMGetDialogDone
    PMSetDialogDone

    PMIsPostScriptDriver
    PMGetLanguageInfo
    PMGetDriverCreator
    PMGetDriverReleaseInfo
    PMGetPrinterResolutionCount
    PMGetPrinterResolution
    PMGetIndexedPrinterResolution
    PMPrinterGetDriverCreator
    PMPrinterGetDriverReleaseInfo
    PMPrinterGetPrinterResolutionCount
    PMPrinterGetPrinterResolution
    PMPrinterGetIndexedPrinterResolution
    PMPrinterGetDescriptionURL
    PMPrinterGetLanguageInfo

    PMBeginDocument
        PMPostScriptBegin
        PMPostScriptEnd
        PMPostScriptHandle
        PMPostScriptData
        PMPostScriptFile

        PMBeginPage
            PMGetGrafPtr
            PMSetProfile
            PMEnableColorSync
            PMDisableColorSync
        PMEndPage
    PMEndDocument
PMEnd
```

Sequence and Scope: Unrestricted Functions

The functions shown in Figure 1-3 have no calling sequence or scope restrictions, and may generally be used anywhere in your printing code.

Figure 1-3 Functions with no calling sequence or scope restrictions

```
PMRetain
PMRelease
PMCreatePageFormat
PMCreatePrintSettings
PMGetJobNameCFString
PMSetJobNameCFString
PMNewPageFormat
PMDisposePageFormat
PMGetDestination
PMCopyPageFormat
PMFlattenPageFormat
PMUnflattenPageFormat
PMGetPageFormatExtendedData
PMSetPageFormatExtendedData
PMGetScale
PMSetScale
PMGetResolution
PMSetResolution
PMGetPhysicalPaperSize
PMSetPhysicalPaperSize
PMGetPhysicalPageSize
PMGetAdjustedPaperRect
PMGetAdjustedPageRect
PMGetUnadjustedPaperRect
PMSetUnadjustedPaperRect
PMGetUnadjustedPageRect
PMSetAdjustedPageRect
PMGetOrientation
PMSetOrientation
PMNewPrintSettings
PMDisposePrintSettings
PMCopyPrintSettings
PMFlattenPrintSettings
PMUnflattenPrintSettings
PMGetPrintSettingsExtendedData
PMSetPrintSettingsExtendedData
PMGetJobName
PMSetJobName
PMGetCopies
PMSetCopies
PMGetFirstPage
PMSetFirstPage
PMGetLastPage
PMSetLastPage
PMGetPageRange
PMSetPageRange
PMGetColorMode
PMSetColorMode
```

C H A P T E R 2

Tasks

This chapter provides information and examples that will help you convert your existing printing code to use the Carbon Printing Manager.

Converting Your Code

Updating your application to use the Carbon Printing Manager is a straightforward process. The basic steps are:

1. Remove all references to the `Printing.h` header file from your project.
2. Add `PMApplication.h` , `PMDefinitions.h`, `PMCore.h`, and the `CarbonLib` library to your project.
3. Convert your code to use the new printing functions and opaque data types, as described in the following sections. You may want to refer to the code examples provided in the appendix. See "Sample Code" (page 147).

New and Changed Printing Functions

Because the Carbon Printing Manager replaces all of the Classic Printing Manager functions you are using today, the first step in converting your code is to locate and replace your Classic Printing Manager function calls with their Carbon equivalents.

In most cases there is a one-to-one mapping between the new functions and the original functions they replace. Table 2-1 lists the Carbon equivalents for Classic Printing Manager functions.

Table 2-1 Carbon replacements for Classic Printing Manager functions

Classic Function	Session Function	Non-Session Function
`PrOpen`	`PMCreateSession`	`PMBegin`
`PrClose`	`PMRelease`	`PMEnd`
`PrOpenDoc`	`PMSessionBeginDocument`	`PMBeginDocument`
`PrCloseDoc`	`PMSessionEndDocument`	`PMEndDocument`
`PrOpenPage`	`PMSessionBeginPage`	`PMBeginPage`
`PrClosePage`	`PMSessionEndPage`	`PMEndPage`
`PrintDefault`	`PMSessionDefaultPrintSettings`	`PMDefaultPrintSettings`
	`PMSessionDefaultPageFormat`	`PMDefaultPageFormat`
`PrValidate`	`PMSessionValidatePrintSettings`	`PMValidatePrintSettings`
	`PMSessionValidatePageFormat`	`PMValidatePageFormat`
`PrJobInit`	`PMSessionPrintDialogInit`	`PMPrintDialogInit`
`PrJobDialog`	`PMSessionPrintDialog`	`PMPrintDialog`
`PrStlInit`	`PMSessionPageSetupDialogInit`	`PMPageSetupDialogInit`
`PrStlDialog`	`PMSessionPageSetupDialog`	`PMPageSetupDialog`
`PrDlgMain`	`PMSessionPrintDialogMain`	`PMPrintDialogMain`
	`PMSessionPageSetupDialogMain`	`PMPageSetupDialogMain`
`PrGeneral`	`PMSessionGeneral`	`PMGeneral`
`PrSetError`	`PMSessionSetError`	`PMSetError`
`PrError`	`PMSessionError`	`PMError`
`MyDoPrintIdle`	`PMIdleProcPtr`	`PMIdleProcPtr`
`MyPrDialogAppend`	`PMPageSetupDialogInitProcPtr`	`PMPageSetupDialogInitProcPtr`
	`PMPrintDialogInitProcPtr`	`PMPrintDialogInitProcPtr`

Some of the functionality provided by Classic Printing Manager is no longer supported. Table 2-2 lists the Classic functions that are not supported by the Carbon Printing Manager.

Table 2-2 Classic Printing Manager functions that are not supported in Carbon

`PrPicFile`
`PrPurge`
`PrNoPurge`
`PrLoadDriver`
`PrDrvrDCE`
`PrDrvrOpen`
`PrDrvrClose`
`PrDrvrVers`
`PrCtlCall`
`PrJobMerge`

The `PrPicFile` function was removed because the "deferred" printing style is no longer supported. All print records must use "draft" style, and printer drivers must perform their own spooling or, on Mac OS 8 and 9, use the Desktop Printer Spooler. Refer to the Classic Printing Manager documentation for information about draft and deferred printing styles. The Desktop Printer Spooler is described in Tech Note 1097.

A direct replacement for the `PrJobMerge` function is unnecessary because the `PMPageFormat` and `PMPrintSettings` objects can be used independently. You can print multiple documents, each with their own saved page format, using a single `PMPrintSettings` object.

New Data Types and Accessor Functions

The Carbon Printing Manager replaces the Classic print record (`TPrint`) with two opaque data types, the `PMPrintSettings` object and the `PMPageFormat` object. All references to elements of the `TPrint` record in your code must be updated to refer to one of these objects, using the new accessor functions provided.

Because the `PMPrintSettings` and `PMPageFormat` objects are opaque, your application must not make assumptions about their size or internal structure. You will need to update any code that loads, stores, or directly manipulates the `TPrint` record.

Table 2-3 lists the accessor functions you can use to examine elements of the `PMPrintSettings` and `PMPageFormat` objects.

Table 2-3 Carbon accessor functions for Classic print record (TPrint) fields

Data Structure	Element	Accessor Function
`TPrint`	`iPrVersion`	not supported (Use `PMGetDriverReleaseInfo` to obtain a driver's version strings.)
	`prInfo`	see `TPrInfo`
	`rPaper`	`PMGetAdjustedPaperRect`
	`prStl`	see `TPrStl`
	`prInfoPT`	not supported
	`prXInfo`	not supported
	`prJob`	see `TPrJob`
	`printX[19]`	not supported
`TPrInfo`	`iDev`	not supported
	`iVRes`	`PMGetResolution`
	`iHRes`	`PMGetResolution`
	`rPage`	`PMGetAdjustedPageRect`

Table 2-3 Carbon accessor functions for Classic print record (TPrint) fields (continued)

Data Structure	Element	Accessor Function
`TPrStl`	`wDev`	`PMGetOrientation`
	`iPageV`	not supported
	`iPageH`	not supported
	`bPort`	not supported
	`feed`	not supported
`TPrJob`	`iFstPage`	`PMGetFirstPage`
	`iLstPage`	`PMGetLastPage`
	`iCopies`	`PMGetCopies`
	`bJDocLoop`	not supported
	`fFromUsr`	not supported
	`pIdleProc`	`PMSetIdleProc`
	`pFileName`	not supported
	`iFileVol`	not supported
	`bFileVers`	not supported
	`bJobX`	not supported

When you use these accessor functions, be sure to pass the appropriate constant for any parameters you do not want to pass to the function or receive from it. For example, pass the `kPMDontWantData` constant in place of a parameter that returns data you're not interested in.

Your application must dispose of any structures, references, or other data returned by Carbon Printing Manager functions. Your application should also dispose of the `PMPrintSettings` and `PMPageFormat` objects when they are no longer needed.

Supported PrGeneral Opcodes

The Carbon Printing Manager provides the `PMSessionGeneral` function (and its non-session equivalent, `PMGeneral`) as a replacement for the Classic `PrGeneral` function. However, Apple suggests that you reduce your reliance on these functions because they are not currently supported by all printer drivers, and because they are not likely to be supported in future versions of the Mac OS.

Table 2-4 lists `PrGeneral` opcodes that are supported in Carbon. For opcodes that have an associated accessor function, you use that function instead of passing the opcode to `PMSessionGeneral`. For example, use `PMGetOrientation` instead of passing the `getRotnOp` constant to `PMSessionGeneral`. The `PMSessionGeneral` function returns the result code `kPMNotImplemented` for any unsupported opcodes, and for opcodes that have Carbon accessor functions.

Table 2-4 Carbon support for PrGeneral opcodes

Opcode	Value	Accessor Function
`GetRslDataOp`	4	`PMPrinterGetIndexedPrinterResolution`
`SetRslOp`	5	`PMSetResolution`
`DraftBitsOp`	6	`PMSessionGeneral` supports this opcode
`NoDraftBitsOp`	7	`PMSessionGeneral` supports this opcode
`getRotnOp`	8	`PMGetOrientation`
`NoGrayScl`	9	`PMSessionGeneral` supports this opcode
`GetPSInfoOp`	10	`PMSessionGeneral` supports this opcode
`PSIntentionsOp`	11	`PMSessionGeneral` supports this opcode
`EnableColorMatchingOp`	12	`PMSessionGeneral` supports this opcode
`PSAdobeOp`	14	`PMSessionGeneral` supports this opcode
`PSPrimaryPPDOp`	15	`PMSessionGeneral` supports this opcode
`kLoadCommProcsOp`	16	`PMSessionGeneral` supports this opcode
`kUnloadCommProcsOp`	17	`PMSessionGeneral` supports this opcode
`kExtendPrintRecOp`	18	`PMSessionGeneral` supports this opcode
`kGetExtendedPrintRecOp`	19	`PMSessionGeneral` supports this opcode

Table 2-4 Carbon support for PrGeneral opcodes (continued)

Opcode	Value	Accessor Function
`kPrinterDirectOpCode`	20	`PMSessionGeneral` supports this opcode
`kSetExtendedPrintRecOp`	21	`PMSessionGeneral` supports this opcode
`kPrVersionOp`	22	`PMSessionGeneral` supports this opcode
`kGetPrinterInfo`	23	`PMSessionGeneral` supports this opcode
`kIsSamePrinterInfo`	24	`PMSessionGeneral` supports this opcode
`kSetDefaultPrinterInfo`	25	`PMSessionGeneral` supports this opcode
`kPrEnablePartialFonts`	26	`PMSessionGeneral` supports this opcode

Supported Picture Comments

Table 2-5 lists picture comments supported by the Carbon Printing Manager.

Table 2-5 Picture comments supported by the Carbon Printing Manager

Name	Value	Data size	Data handle	Description
Text picture comments				
`TextBegin`	150	6	`TTxtPicRec`	Begin text function
`TextEnd`	151	0	`NULL`	End text function
`StringBegin`	152	0	`NULL`	Begin string delimitation
`StringEnd`	153	0	`NULL`	End string delimitation
`TextCenter`	154	8	`TCenterRec`	Offset to center of rotation for text
`LineLayoutOff`	155	0	`NULL`	Turn printer driver's line layout off
`LineLayoutOn`	156	0	`NULL`	Turn printer driver's line layout on
`ClientLineLayout`	157	16	`TClientLLRec`	Customize line layout error distribution

Table 2-5 Picture comments supported by the Carbon Printing Manager (continued)

Name	Value	Data size	Data handle	Description
Graphics picture comments				
`PolyBegin`	160	0	`NULL`	Begin special polygon
`PolyEnd`	161	0	`NULL`	End special polygon
`PolyIgnore`	163	0	`NULL`	Ignore following polygon data
`PolySmooth`	164	1	`TPolyVerbRec`	Close, fill, frame
`PolyClose`	165	0	`NULL`	Smooth the curve between endpoints
`RotateBegin`	200	8	`TRotationRec`	Begin rotated port
`RotateEnd`	201	0	`NULL`	End rotation
`RotateCenter`	202	8	`TCenterRec`	Offset to center of rotation
Line-drawing picture comments				
`DashedLine`	180	Size of a `TDashedLineRec` record	`TDashedLineRec`	Draw following line as dashed
`DashedStop`	181	0	`NULL`	End dashed lines
`SetLineWidth`	182	4	`TLineWidthHdl`	Set fractional line widths
PostScript picture comments				
`PostScriptBegin`	190	0	`NULL`	Set driver state to PostScript
`PostScriptEnd`	191	0	`NULL`	Restore QuickDraw state
`PostScriptHandle`	192	Size of the PostScript data in handle	a handle	PostScript data referenced by handle
`PostScriptFile`	193	Size of the PostScript data in handle	a handle	Filename referenced by handle

Table 2-5 Picture comments supported by the Carbon Printing Manager (continued)

Name	Value	Data size	Data handle	Description
`TextIsPostScript`	194	0	`NULL`	QuickDraw text is sent as PostScript
`ResourcePS`	195	8	Resource type, resource ID	Index PostScript data in a resource file
`PSBeginNoSave`	196	0	`NULL`	Set driver state to PostScript
ColorSync picture comments				
`CMBeginProfile`	220	0	`NULL`	Begin ColorSync profile
`CMEndProfile`	221	0	`NULL`	End ColorSync profile
`CMEnableMatching`	222	0	`NULL`	Begin ColorSync color matching
`CMDisableMatching`	223	0	`NULL`	End ColorSync color matching

Saving a Page Format Object in a Document

Listing 2-1 shows an example of how you can save a `PMPageFormat` object in a document.

Listing 2-1 Saving a `PMPageFormat` object

```
PMPrintSettings  mySettings;
PMPageFormat     myFormat;
Ptr              myData;
UInt32           myDataSize;
Handle           flattenFormat;

PMBegin();
    GetMyData ( &myData, &myDataSize );
    PMSetPageFormatExtendedData ( myFormat, kMyDataID,
                                 myDataSize, (void*) myData );
    PMFlattenPageFormat ( myFormat, &flattenFormat );
    /* Attach the flattened format handle to the document and save it. */
    DisposeHandle ( flattenFormat );
PMEnd();
```

Retrieving a Saved Page Format Object

Listing 2-2 shows an example of how you can retrieve a `PMPageFormat` object that was saved with a document.

Listing 2-2 Retrieving a `PMPageFormat` object

```
PMPrintSettings  mySettings;
PMPageFormat     myFormat;
UInt32           myDataSize;
Handle           flattenFormat;
Boolean          changed;
Ptr              myData;

PMBegin();
    PMUnflattenPageFormat ( flattenFormat, &myFormat );
    PMValidatePageFormat ( myFormat, &changed );
    /* First get the size of the data block so */
    /* we know how much storage to allocate. * /
    PMGetPageFormatExtendedData ( myFormat, kMyDataID,
                                 &myDataSize, kPMDontWantData );
    myData = NewPtr( myDataSize );
    PMGetPageFormatExtendedData ( myFormat, kMyDataID,
                                 &myDataSize, (void*) myData);
PMEnd();
```

Reference

Functions

Managing Printing Objects

`PMRetain`

Increments a printing object's reference count.

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

`object`	A Carbon Printing Manager object, such as a `PMPrintSession`, `PMPageFormat`, or `PMPrintSettings` object.
function result	A result code. See "Result Codes" (page 144).

Discussion
You can use the `PMRetain` function to increment a printing object's reference count, so that multiple threads can use the object without the risk of another thread deallocating the object.s

Special Considerations
Valid after creating a printing object.

PMRelease

Decrements a printing object's reference count.

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

`object`
: A Carbon Printing Manager object, such as a `PMPrintSession`, `PMPageFormat`, or `PMPrintSettings` object.

function result A result code. See "Result Codes" (page 144).

Discussion

Your application should use the `PMRelease` function to release any printing objects it creates. When an object's reference count reaches 0, the object is deallocated.

To terminate a printing session created with `PMCreateSession`, pass the associated `PMPrintSession` object to `PMRelease`.

Special Considerations

Valid after creating a printing object.

Carbon Porting Notes

`PMRelease` is equivalent to the Classic Printing Manager function `PrClose` when the parameter is a `PMPrintSession` object and its reference count is zero.

Print Loop Functions

PMCreateSession

Initializes a `PMPrintSession` object and creates a context for printing operations.

```
OSStatus PMCreateSession (PMPrintSession* printSession);
```

`printSession`
: A `PMPrintSession` object.

function result A result code. See "Result Codes" (page 144).

Discussion

The `PMCreateSession` function creates a context for printing operations. This context is called a printing session.

On Mac OS X, you can use `PMCreateSession` to implement multithreaded printing, and you can create multiple sessions within a single-threaded application. On Mac OS 8 and 9, an application is limited to a single printing session using one thread. An error is returned if you try to nest calls to `PMCreateSession` within a single thread, or to create multiple sessions with separate threads.

Note that each printing session is limited to one dialog box at a time. Each printing session can have its own dialog box, and settings changed in one dialog box are independent of settings in any other dialog box. Because certain calls will block, it is not possible to bring up either the Page Setup or Print dialog boxes from different sessions at the same time in a single thread.

Special Considerations

You must call `PMCreateSession` before any other Carbon Printing Manager function.

Carbon Porting Notes

`PMCreateSession` is equivalent to the Classic Printing Manager function `PrOpen`.

PMSessionBeginDocument

Establishes a graphics context for imaging a document in the specified printing session.

```
OSStatus PMSessionBeginDocument (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat pageFormat);
```

`printSession`	A `PMPrintSession` object.
`printSettings`	A `PMPrintSettings` object.
`pageFormat`	A `PMPageFormat` object.
function result	A result code. See “Result Codes” (page 144).

Special Considerations

Valid within the context of a printing session.

Carbon Porting Notes

`PMSessionBeginDocument` is equivalent to the Classic Printing Manager function `PrOpenDoc`.

PMSessionEndDocument

Closes the context created for imaging a document in the specified printing session.

```
OSStatus PMSessionEndDocument (PMPrintSession printSession);
```

`printSession`
: A `PMPrintSession` object. On return, the object is no longer valid; however, you must still call `PMRelease` to release the object.

function result A result code. See "Result Codes" (page 144).

Discussion

You should call `PMSessionEndDocument` after the last `PMSessionEndPage` in your print loop and before releasing the session.

Special Considerations

Valid within the context of a printing session, after calling `PMSessionBeginDocument`.

Carbon Porting Notes

`PMSessionEndDocument` is equivalent to the Classic Printing Manager function `PrCloseDoc`.

PMSessionBeginPage

Initializes a scaling rectangle for printing a page in the specified printing session.

```
OSStatus PMSessionBeginPage (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   const PMRect* pageFrame);
```

`printSession`
: A `PMPrintSession` object.

`pageFormat`
: A `PMPageFormat` object.

`pageFrame`
: A pointer to a bounding rectangle for drawing the page. This value is usually obtained from the `PMGetAdjustedPageRect` (page 69) function, but if no scaling is needed, pass `NULL`. On Mac OS 8 and 9, this parameter is passed to the printer driver. Classic printer drivers typically scale the contents of the page to fit the specified rectangle. This parameter is ignored when running on Mac OS X.

function result A result code. See "Result Codes" (page 144).

Discussion

Because `PMSessionBeginPage` also initializes the printing port, your application should not make assumptions about the state of the port (for example, the current font) between successive pages. After each call to `PMSessionBeginPage`, your application should call `PMSessionGetGraphicsContext` (page 87) to obtain the current printing port, and set the drawing port to this port using the QuickDraw `SetPort` function.

Special Considerations

Valid within the context of a printing session after calling `PMSessionBeginDocument`.

Carbon Porting Notes

`PMSessionBeginPage` is equivalent to the Classic Printing Manager function `PrOpenPage`.

PMSessionEndPage

Finishes printing the current page.

```
OSStatus PMSessionEndPage (PMPrintSession printSession);
```

`printSession`
A `PMPrintSession` object.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session, after calling `PMSessionBeginPage` and before calling `PMSessionEndDocument`.

Carbon Porting Notes

`PMSessionEndPage` is equivalent to the Classic Printing Manager function `PrClosePage`.

PMSessionSetIdleProc

Installs an idle callback function in your print loop.

```
OSStatus PMSessionSetIdleProc (
   PMPrintSession printSession,
   PMIdleUPP idleProc);
```

`printSession`
: A `PMPrintSession` object.

`idleProc`
: A universal procedure pointer to your idle function. Your idle function is defined by the callback `PMIdleProcPtr` (page 126).

function result A result code. See "Result Codes" (page 144).

Discussion

The Carbon Printing Manager calls your idle function periodically during your print loop. Note that your idle function should not be used to display printing progess information, as that is handled automatically by the Carbon Printing Manager.

Special Considerations

Valid within the context of a printing session.

Carbon Porting Notes

`PMSessionSetIdleProc` is equivalent to setting the `TPrJob.pIdleProc` field in the Classic print record (`TPrint`).

Creating and Using Page Format and Print Settings Objects

PMCreatePageFormat

Creates a new `PMPageFormat` object.

```
OSStatus PMCreatePageFormat (PMPageFormat* pageFormat);
```

`pageFormat`
: On return, a initialized `PMPageFormat` object.

function result A result code. See "Result Codes" (page 144).

Discussion

`PMCreatePageFormat` allocates memory for a new `PMPageFormat` object in your application's memory space. The new page format object is empty until you set its values, or until you call `PMSessionDefaultPageFormat` (page 39) or `PMSessionValidatePageFormat` (page 39).

Carbon Porting Notes

PMCreatePageFormat combined with PMCreatePrintSettings is equivalent to creating a new Classic print record (TPrint).

PMSessionDefaultPageFormat

Assigns default parameter values to a PMPageFormat object for the specified printing session.

```
OSStatus PMSessionDefaultPageFormat (
   PMPrintSession printSession,
   PMPageFormat pageFormat);
```

printSession	A PMPrintSession object.
pageFormat	A PMPageFormat object. On return, the object contains default parameter values.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session after creating a page format object.

Carbon Porting Notes

PMSessionDefaultPageFormat combined with PMSessionDefaultPrintSettings is equivalent to the Classic Printing Manager function PrintDefault.

PMSessionValidatePageFormat

Obtains a PMPageFormat object that is valid within the context of the specified printing session.

```
OSStatus PMSessionValidatePageFormat (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   Boolean* result);
```

printSession	A PMPrintSession object.
pageFormat	The PMPageFormat object to be validated.

`result`
: On return, a value of `true` if any parameters were changed, or `false` if no changes were required.

function result A result code. See “Result Codes” (page 144).

Special Considerations

Valid after creating a page format object with `PMCreatePageFormat` or `PMSessionConvertOldPrintRecord`.

Carbon Porting Notes

`PMSessionValidatePageFormat` combined with `PMSessionValidatePrintSettings` is equivalent to the Classic Printing Manager function `PrValidate`.

PMCreatePrintSettings

Creates a new `PMPrintSettings` object.

```
OSStatus PMCreatePrintSettings (
   PMPrintSettings* printSettings);
```

`printSettings`
: On return, an initialized `PMPrintSettings` object.

function result A result code. See “Result Codes” (page 144).

Discussion

`PMCreatePrintSettings` allocates memory for a new `PMPrintSettings` object in your application’s memory space. The new print settings object is empty until you set its values, or until you call `PMSessionDefaultPrintSettings` (page 40) or `PMSessionValidatePrintSettings` (page 41).

Carbon Porting Notes

`PMCreatePrintSettings` combined with `PMCreatePageFormat` is equivalent to creating a new Classic Printing Record (`TPrint`).

PMSessionDefaultPrintSettings

Assigns default parameter values to a `PMPrintSettings` object for the specified printing session.

```
OSStatus PMSessionDefaultPrintSettings (
   PMPrintSession printSession,
   PMPrintSettings printSettings);
```

`printSession`
: A `PMPrintSession` object.

`printSettings`
: A `PMPrintSettings` object. On return, the object contains default parameter values.

function result A result code. See "Result Codes" (page 144).

Special Considerations
Valid within the context of a printing session after creating a print settings object.

Carbon Porting Notes
`PMSessionDefaultPrintSettings` combined with `PMSessionDefaultPageFormat` is equivalent to the Classic Printing Manager function `PrintDefault`.

PMSessionValidatePrintSettings

Obtains a `PMPrintSettings` object that is valid within the context of the specified printing session.

```
OSStatus PMSessionValidatePrintSettings (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   Boolean* result);
```

`printSession`
: A `PMPrintSession` object.

`printSettings`
: The `PMPrintSettings` object to be validated.

`result`
: On return, a value of `true` if any parameters were changed, or `false` if no changes were required.

function result A result code. See "Result Codes" (page 144).

Special Considerations
Valid after creating a print settings object with `PMCreatePrintSettings` or `PMSessionConvertOldPrintRecord`.

Carbon Porting Notes
`PMSessionValidatePrintSettings` combined with `PMSessionValidatePageFormat` is equivalent to the Classic Printing Manager function `PrValidate`.

Saving and Restoring Page Format and Print Settings Objects

PMFlattenPageFormat

Flattens a PMPageFormat object for storage in a user document.

```
OSStatus PMFlattenPageFormat (
   PMPageFormat pageFormat,
   Handle* flatFormat);
```

pageFormat	A PMPageFormat object.
flatFormat	On return, a handle to a flattened PMPageFormat object.
function result	A result code. See “Result Codes” (page 144).

Discussion

You are responsible for disposing of the handle to the flattened page format object.

Special Considerations

Unlike most Carbon Printing Manager functions, you may call this function outside of a printing session or PMBegin/PMEnd block.

PMUnflattenPageFormat

Rebuilds a flattened PMPageFormat object.

```
OSStatus PMUnflattenPageFormat (
   Handle flatFormat,
   PMPageFormat* pageFormat);
```

flatformat	A handle to a previously flattened PMPageFormat object.
pageFormat	A pointer to the address where the newly created PMPageFormat object should be stored. On return, this object contains the data retrieved from the flattened page format.
function result	A result code. See “Result Codes” (page 144).

Discussion
The `PMUnflattenPageFormat` function creates a new `PMPageFormat` object containing the information from the flattened page format. You are responsible for disposing of the handle to the flattened page format.

Special Considerations
Unlike most Carbon Printing Manager functions, you may call this function outside of a printing session or `PMBegin`/`PMEnd` block.

PMFlattenPrintSettings

Flattens a `PMPrintSettings` object for storage in a user document.

```
OSStatus PMFlattenPrintSettings (
   PMPrintSettings printSettings,
   Handle* flatSettings);
```

`printSettings`	A `PMPrintSettings` object.
`flatSettings`	On return, a handle to a flattened `PMPrintSettings` object.
function result	A result code. See "Result Codes" (page 144).

Discussion
You are responsible for disposing of the handle to the flattened print settings object.

Special Considerations
Unlike most Carbon Printing Manager functions, you may call this function outside of a printing session or `PMBegin`/`PMEnd` block.

PMUnflattenPrintSettings

Rebuilds a flattened `PMPrintSettings` object.

```
OSStatus PMUnflattenPrintSettings (
   Handle flatSettings,
   PMPrintSettings* printSettings);
```

`flatSettings`	A handle to a previously flattened `PMPrintSettings` object. Your application must dispose of this handle.

`printSettings`
: A pointer to the address where the newly created `PMPringSettings` object should be stored. On return, this object contains the data retrieved from the flattened print settings.

function result
: A result code. See "Result Codes" (page 144). The result code `kPMInvalidParameter` is returned if the flattened `PMPrintSettings` object was created by an incompatible version of the Carbon Printing Manager. In this case your application should display the Print dialog box to allow the user to enter the desired settings.

Discussion

The `PMUnflattenPrintSettings` function creates a new `PMPrintSettings` object containing the information from the flattened print settings.

Special Considerations

Unlike most Carbon Printing Manager functions, you may call this function outside of a printing session or `PMBegin`/`PMEnd` block.

Making Copies of Page Format and Print Settings Objects

PMCopyPageFormat

Copies the settings from one `PMPageFormat` object into another.

```
OSStatus PMCopyPageFormat (
   PMPageFormat formatSrc,
   PMPageFormat formatDest);
```

`formatSrc`
: A `PMPageFormat` object to be duplicated.

`formatDest`
: On return, a `PMPageFormat` object containing the same settings as the `formatSrc` object.

function result
: A result code. See "Result Codes" (page 144).

Discussion

You must create both page format objects before calling this function.

Special Considerations

Unlike most Carbon Printing Manager functions, you may call this function outside of a printing session or `PMBegin`/`PMEnd` block.

PMCopyPrintSettings

Copies the settings from one `PMPrintSettings` object into another.

```
OSStatus PMCopyPrintSettings (
   PMPrintSettings settingSrc,
   PMPrintSettings settingDest);
```

`settingSrc`	A `PMPrintSettings` object to be duplicated.
`settingDest`	On return, a `PMPrintSettings` object containing the same settings as the `settingSrc` object.
function result	A result code. See “Result Codes” (page 144).

Discussion

You must create both print settings objects before calling this function.

Special Considerations

Unlike most Carbon Printing Manager functions, you may call this function outside of a printing session or `PMBegin`/`PMEnd` block.

Extending the Page Format and Print Settings Objects

PMGetPageFormatExtendedData

Obtains extended page format data previously stored by your application.

```
OSStatus PMGetPageFormatExtendedData (
   PMPageFormat pageFormat,
   OSType dataID,
   UInt32* size,
   void* extendedData);
```

`pageFormat`	A `PMPageFormat` object.

`dataID`
: The unique 4-character identifier of the data to retrieve. This is typically your application's creator code.

`size`
: A pointer to a variable specifying the size of the buffer you have allocated for the extended page format data. On return, this variable contains the number of bytes read into the buffer, or the size of the extended data. You can pass the constant `kPMDontWantSize` if you do not need this information.

`extendedData`
: A pointer to a buffer to receive the extended data. Pass the constant `kPMDontWantData` if you do not want to read the data.

function result A result code. See "Result Codes" (page 144).

Discussion

Your application will typically need to call `PMGetPageFormatExtendedData` two times in order to retrieve the extended page format data. The first time, pass the constant `kPMDontWantData` in the `extendedData` field, to obtain the buffer size required for the extended data. Then call the function a second time to read the extended data into your buffer.

Special Considerations

Before using this function you must call `PMSessionValidatePageFormat` (page 39) or `PMValidatePageFormat` (page 100) to ensure that the page format object is valid.

Valid within a printing session after creating a page format object.

PMSetPageFormatExtendedData

Stores your application-specific data in a page format object.

```
OSStatus PMSetPageFormatExtendedData (
   PMPageFormat pageFormat,
   OSType dataID,
   UInt32 size,
   void* extendedData);
```

`pageFormat`
: A `PMPageFormat` object.

`dataID`
A 4-character code that will be used to identify your data. This is typically your application's creator code.

`size`
The size, in bytes, of the data to be stored the page format object.

`extendedData`
A pointer to the data to be stored.

function result A result code. See "Result Codes" (page 144).

Special Considerations
Valid within a printing session after creating a page format object.

PMGetPrintSettingsExtendedData

Obtains extended print settings data previously stored by your application.

```
OSStatus PMGetPrintSettingsExtendedData (
   PMPrintSettings printSettings,
   OSType dataID,
   UInt32* size,
   void* extendedData);
```

`printSettings`
A `PMPrintSettings` object.

`dataID`
The unique 4-character identifier of the data to retrieve. This is typically your application's creator code.

`size`
A pointer to a variable specifying the size of the buffer you have allocated for the extended print settings data. On return, this variable contains the number of bytes read into the buffer, or the size of the extended data. You can pass the constant `kPMDontWantSize` if you do not need this information.

`extendedData`
A pointer to a buffer to receive the extended data. Pass the constant `kPMDontWantData` if you do not want to read the data.

function result A result code. See "Result Codes" (page 144).

Discussion

Your application will typically need to call `PMGetPrintSettingsExtendedData` two times in order to retrieve the extended print settings data. The first time, pass the constant `kPMDontWantData` in the `extendedData` field, to obtain the buffer size required for the extended data. Then call the function a second time to read the extended data into your buffer.

Special Considerations

Before using this function you must call `PMSessionValidatePrintSettings` (page 41) or `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid within a printing session after creating a print settings object.

PMSetPrintSettingsExtendedData

Stores your application-specific data in a `PMPrintSettings` object.

```
OSStatus PMSetPrintSettingsExtendedData (
   PMPrintSettings printSettings,
   OSType dataID,
   UInt32 size,
   void* extendedData);
```

`printSettings`
: A `PMPrintSettings` object.

`dataID`
: A 4-character code that will be used to identify your data. This is typically your application's creator code.

`size`
: The size, in bytes, of the data to be stored the print settings object.

`extendedData`
: A pointer to the data to be stored.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within a printing session after creating a print settings object.

Print Record Compatibility Functions

PMSessionConvertOldPrintRecord

Creates new PMPageFormat and PMPrintSettings objects from a Classic print record.

```
OSStatus PMSessionConvertOldPrintRecord (
   PMPrintSession printSession,
   Handle printRecordHandle,
   PMPrintSettings* printSettings,
   PMPageFormat* pageFormat);
```

printSession	A PMPrintSession object.
printRecordHandle	A handle to a Classic print record.
printSettings	On return, a validated PMPrintSettings object.
pageFormat	On return, a validated PMPageFormat object.
function result	A result code. See “Result Codes” (page 144).

Discussion
Use PMSessionConvertOldPrintRecord to create page format and print settings objects from documents created by non-Carbon versions of your application.

Special Considerations
Valid within the context of a printing session.

PMSessionMakeOldPrintRecord

Creates a Classic print record from a PMPageFormat and a PMPrintSettings object.

```
OSStatus PMSessionMakeOldPrintRecord (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat pageFormat,
   Handle* printRecordHandle);
```

printSession	A PMPrintSession object.

`printSettings`
: A `PMPrintSettings` object.

`pageFormat`
: A `PMPageFormat` object.

`printRecordHandle`
: On return, a handle to a Classic print record. Your application must dispose of this handle.

function result A result code. See "Result Codes" (page 144).

Discussion
Use `PMSessionMakeOldPrintRecord` to create a Classic print record to store with your documents for compatibility with non-Carbon versions of your application. Note that because the page format and print settings objects contain more information than the Classic print record, some settings may be lost in conversion.

Special Considerations
Valid within a printing session after creating a page format object and a print settings object.

Displaying the Page Setup and Print Dialog Boxes

PMSessionPageSetupDialog

Displays the Page Setup dialog box and records the user's selections in a `PMPageFormat` object.

```
OSStatus PMSessionPageSetupDialog (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   Boolean* accepted);
```

`printSession`
: A `PMPrintSession` object.

`pageFormat`
: A `PMPageFormat` object.

`accepted`
: Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.

function result A result code. See "Result Codes" (page 144).

Discussion

When the Page Setup dialog box is displayed, it shows the current settings contained in the page format object. If the user changes these settings and clicks the OK button, the page format object is updated with the user's selections. If the user clicks the Cancel button, the page format object is returned unchanged.

Special Considerations

Valid within the context of a printing session. Never call `PMSessionPageSetupDialog` between the pages of a document.

Carbon Porting Notes

`PMSessionPageSetupDialog` is equivalent to the Classic Printing Manager function `PrStlDialog`.

PMSessionPrintDialog

Displays the Print dialog box and records the user's selections in a `PMPrintSettings` object.

```
OSStatus PMSessionPrintDialog (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   Boolean* accepted);
```

`printSession`
A `PMPrintSession` object.

`printSettings`
A `PMPrintSettings` object.

`constPageFormat`
A `PMPageFormat` object.

`accepted`
Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.

function result A result code. See "Result Codes" (page 144).

Discussion

When the Print dialog box is displayed, it shows the current settings in the print settings object. If the user changes these settings and clicks the OK button, the print settings object is updated with the user's selections. If the user clicks the Cancel button, the print settings object is returned unchanged.

Special Considerations

Valid within the context of a printing session.

Carbon Porting Notes

`PMSessionPrintDialog` is equivalent to the Classic Printing Manager function `PrJobDialog`.

Customizing the Page Setup and Print Dialog Boxes

PMSessionPageSetupDialogInit

Initializes a custom Page Setup dialog box.

```
OSStatus PMSessionPageSetupDialogInit (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   PMDialog* newDialog);
```

`printSession`
: A `PMPrintSession` object.

`pageFormat`
: A `PMPageFormat` object.

`newDialog`
: On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMSessionPageSetupDialogMain` (page 53) function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPageFormat` object. See the discussion of the `PMPageSetupDialogInitProcPtr` (page 127) callback function for more information.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session after creating a page format object.

Carbon Porting Notes

`PMSessionPageSetupDialogInit` is equivalent to the Classic Printing Manager function `PrStlInit`.

PMSessionPageSetupDialogMain

Displays your application's custom Page Setup dialog box.

```
OSStatus PMSessionPageSetupDialogMain (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   Boolean* accepted,
   PMPageSetupDialogInitUPP myInitProc);
```

`printSession`	A `PMPrintSession` object.
`pageFormat`	A `PMPageFormat` object.
`accepted`	Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.
`myInitProc`	A universal procedure pointer to your dialog initialization function. Your initialization function is defined by the callback `PMPageSetupDialogInitProcPtr` (page 127).
function result	A result code. See "Result Codes" (page 144).

Discussion

The Carbon Printing Manager calls your dialog initialization function before displaying your custom Page Setup dialog box. Your initialization function can append items to the Page Setup dialog box, and should install an event filter using the `PMSetModalFilterProc` (page 56) function. You must pass the same `PMPageFormat` and `PMPrintSession` objects to both `PMSessionPageSetupDialogMain` and `PMSessionPageSetupDialogInit` (page 52).

Special Considerations

Valid within the context of a printing session after calling `PMSessionPageSetupDialogInit`.

Carbon Porting Notes

`PMSessionPageSetupDialogMain` is equivalent to the Classic Printing Manager function `PrDlgMain`.

PMSessionPrintDialogInit

Initializes a custom Print dialog box.

```
OSStatus PMSessionPrintDialogInit (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   PMDialog* newDialog);
```

`printSession`	A `PMPrintSession` object.
`printSettings`	A `PMPrintSettings` object.
`constPageFormat`	A `PMPageFormat` object.
`newDialog`	On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMSessionPrintDialogMain` (page 55) function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPrintSettings` object. See the discussion of the `PMPrintDialogInitProcPtr` (page 129) callback function for more information.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session after creating a page format object.

Carbon Porting Notes

`PMSessionPrintDialog` is equivalent to the Classic Printing Manager function `PrJobInit`.

PMSessionPrintDialogMain

Displays your application's custom Print dialog box.

```
OSStatus PMSessionPrintDialogMain (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   Boolean* accepted,
   PMPrintDialogInitUPP myInitProc);
```

`printSession`	A `PMPrintSession` object.
`printSettings`	A `PMPrintSettings` object.
`constPageFormat`	A `PMPageFormat` object.
`accepted`	Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.
`myInitProc`	A universal procedure pointer to your dialog initialization function. Your initialization function is defined by the callback `PMPrintDialogInitProcPtr` (page 129).
function result	A result code. See "Result Codes" (page 144).

Discussion

The Carbon Printing Manager calls your dialog initialization function before displaying your custom Print dialog box. Your initialization function can append items to the Print dialog box, and should install an event filter using the `PMSetModalFilterProc` (page 56) function. You must pass the same `PMPageFormat` and `PMPrintSession` objects to both `PMSessionPrintDialogMain` and `PMSessionPrintDialogInit` (page 54).

Special Considerations

Valid within the context of a printing session after calling `PMSessionPrintDialogInit`.

Carbon Porting Notes

`PMSessionPrintDialogMain` is equivalent to the Classic Printing Manager function `PrDlgMain`.

PMGetModalFilterProc

Obtains the event handling callback function for a modal dialog box.

```
OSStatus PMGetModalFilterProc (
   PMDialog pmDialog,
   ModalFilterUPP* filterProc);
```

`pmDialog`	A `PMDialog` object representing the Page Setup or Print dialog box for which you want to obtain the event handling function.
`filterProc`	On return, a pointer to the dialog event handling function.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session after installing an event handling function for your Page Setup or Print dialog box using the `PMSetModalFilterProc` function.

Carbon Porting Notes

`PMGetModalFilterProc` is equivalent to setting the `TPrDlg.pFltrProc` field in the Classic Printing Manager.

PMSetModalFilterProc

Installs an event handling callback function for a modal dialog box.

```
OSStatus PMSetModalFilterProc (
   PMDialog pmDialog,
   ModalFilterUPP filterProc);
```

`pmDialog`	A `PMDialog` object representing the Page Setup or Print dialog box for which you want to install an event handling function.
`filterProc`	Your event handling function, which will be called while the Page Setup or Print dialog box is displayed.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session. You must register your filter procedure before calling `PMSessionPageSetupDialogMain` or `PMSessionPrintDialogMain`.

Carbon Porting Notes

`PMSetModalFilterProc` is equivalent to setting the `TPrDlg.pFltrProc` field in the Classic Printing Manager.

PMGetItemProc

Obtains the event handling callback function for a custom dialog box.

```
OSStatus PMGetItemProc (
   PMDialog pmDialog,
   PMItemUPP* itemProc);
```

`pmDialog`	A `PMDialog` object representing the Page Setup or Print dialog box for which you want to obtain the event handling function.
`itemProc`	On return, a pointer to the dialog event handling function.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session after calling `PMSetItemProc` to install an event handling function for your custom Page Setup or Print dialog box.

Carbon Porting Notes

`PMGetItemProc` is equivalent to setting the `TPrDlg.pItemProc` field in the Classic Printing Manager.

PMSetItemProc

Installs an event handling callback function for items in a custom dialog box.

```
OSStatus PMSetItemProc (
   PMDialog pmDialog,
   PMItemUPP itemProc);
```

`pmDialog`
: A `PMDialog` object representing your custom Page Setup or Print dialog box.

`itemProc`
: A universal procedure pointer to your dialog item event handling function. Your event handling function is defined by the callback `PMItemProcPtr` (page 130).

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session after creating a custom Page Setup or Print dialog box.

Carbon Porting Notes

`PMSetItemProc` is equivalent to installing a procedure pointer in the Classic Printing Manager's `TPrDlg.pItemProc` field.

PMGetDialogPtr

Obtains the dialog record structure for a dialog box.

```
OSStatus PMGetDialogPtr  (
   PMDialog pmDialog,
   DialogPtr* theDialog);
```

`pmDialog`
: The `PMDialog` object from which you wish to extract the `DialogRecord` structure.

`theDialog`
: On return, a pointer to a Dialog Manager structure of type `DialogRecord`.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session after calling `PMSessionPageSetupDialogInit` to initialize a Page Setup dialog box, or calling `PMSessionPrintDialogInit` to initialize a Print dialog box.

Carbon Porting Notes

`PMGetDialogPtr` is equivalent to reading the Classic Printing Manager `TPrDlg.Dlg` field.

PMGetDialogAccepted

Reports to application-defined callback functions whether the user has confirmed a dialog box.

```
OSStatus PMGetDialogAccepted (
   PMDialog pmDialog,
   Boolean* process);
```

`pmDialog`
: A `PMDialog` object representing your customized Page Setup or Print dialog box.

`process`
: Returns `true` if the user confirms the dialog box. In the case of the Page Setup dialog box, this means the user clicked the OK button; in the case of the Print dialog box, this means the user clicked the Print button. Returns `false` if the user clicks the Cancel button.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session while displaying your Page Setup or Print dialog box.

Carbon Porting Notes

`PMGetDialogAccepted` corresponds to checking the Classic Printing Manager `TPrDlg.fDoIt` field.

PMSetDialogAccepted

Informs the Carbon Printing Manager that the user confirmed a custom dialog box.

```
OSStatus PMSetDialogAccepted (
   PMDialog pmDialog,
   Boolean process);
```

`pmDialog`
: A `PMDialog` object representing your customized Page Setup or Print dialog box.

`process`

Pass `true` if the user confirms the dialog box. In the case of the Page Setup dialog box, this means the user clicked the OK button; in the case of the Print dialog box, this means the user clicked the Print button. Pass `false` if the user clicks the Cancel button.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session while displaying your Page Setup or Print dialog box.

Carbon Porting Notes

`PMSetDialogAccepted` corresponds to setting the Classic Printing Manager `TPrDlg.fDoIt` field.

PMGetDialogDone

Reports to application-defined callback functions whether the user has finished with a dialog box.

```
OSStatus PMGetDialogDone (
   PMDialog pmDialog,
   Boolean* done);
```

`pmDialog`

A `PMDialog` object representing your customized Page Setup or Print dialog box.

`done`

Returns `true` if the user clicked the OK, Print, or Cancel button; `false` if the user did not click any of these buttons.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session while displaying your Page Setup or Print dialog.

Carbon Porting Notes

`PMGetDialogDone` corresponds to checking the Classic Printing Manager `TPrDlg.fDone` field.

PMSetDialogDone

Informs the Carbon Printing Manager that the user finished with a custom dialog box.

```
OSStatus PMSetDialogDone (
   PMDialog pmDialog,
   Boolean done);
```

`pmDialog`
: A `PMDialog` object representing your custom Page Setup or Print dialog box.

`done`
: Pass `true` when your callback function has finished with this dialog box.

function result A result code. See “Result Codes” (page 144).

Special Considerations

Valid within the context of a printing session while displaying your Page Setup or Print dialog box.

Carbon Porting Notes

`PMSetDialogDone` corresponds to setting the Classic Printing Manager `TPrDlg.fDone` field.

Getting Information About a Printer

PMPrinterGetDescriptionURL

Obtains a reference to the specified printer’s description file, typically a PostScript Printer Description (PPD) file for a PostScript printer.

```
OSStatus PMPrinterGetDescriptionURL (
   PMPrinter printer,
   CFStringRef descriptionType,
   CFURLRef* fileURL);
```

`printer`
: A `PMPrinter` object.

`descriptionType`
: The type of description file for the selected printer.

`fileURL`
: On return, the location of the printer description file.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session.

PMSessionGetCurrentPrinter

Creates a new `PMPrinter` object for the current printer, associated with the specified printing session.

```
OSStatus PMSessionGetCurrentPrinter (
   PMPrintSession printSession,
   PMPrinter* currentPrinter);
```

`printSession`
: A `PMPrintSession` object.

`currentPrinter`
: On return, a new `PMPrinter` object.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session.

PMPrinterGetLanguageInfo

Obtains information about the specified printer's imaging language.

```
OSStatus PMPrinterGetLanguageInfo (
   PMPrinter printer,
   PMLanguageInfo* info);
```

`printer`
: A `PMPrinter` object.

`info`
: On return, a pointer to a data structure containing the printer's language level, version and release. The format of the returned data is based on the PostScript language.

function result A result code. See "Result Codes" (page 144).

Discussion

`PMPrinterGetLanguageInfo` is useful for PostScript printers, but may be irrelevant for other types of printers.

Special Considerations

Valid within the context of a printing session.

`PMPrinterGetDriverCreator`

Obtains the creator of the driver associated with the specified printer.

```
OSStatus PMPrinterGetDriverCreator (
   PMPrinter printer,
   OSType* creator);
```

`printer`
: A `PMPrinter` object.

`creator`
: On return, the 4-byte creator type of the driver (for example, `'APPL'` for an Apple printer driver).

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session.

Carbon Porting Notes

For Classic printer drivers that support the `PrGeneral` opcode `kPrinterDirectOpCode`, `PMPrinterGetDriverCreator` provides functionality equivalent to calling `PrGeneral`, passing in the `kPDVerifySelect` selector to obtain the driver's creator type and version.

PMPrinterGetDriverReleaseInfo

Obtains release information for the driver associated with the specified printer.

```
OSStatus PMPrinterGetDriverReleaseInfo (
   PMPrinter printer,
   VersRec* release);
```

`printer`
: A `PMPrinter` object.

`release`
: On return, a pointer to a `VersRec` data structure containing the driver's short and long version strings and country code.

function result
: A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session.

Carbon Porting Notes

For Classic printer drivers that support the `PrGeneral` opcode `kPrinterDirectOpCode`, `PMPrinterGetDriverReleaseInfo` provides functionality equivalent to calling `PrGeneral`, passing in the `kPDVerifySelect` selector to obtain the driver's creator type and version.

PMPrinterGetPrinterResolutionCount

Obtains the number of resolution settings supported by the specified printer.

```
OSStatus PMPrinterGetPrinterResolutionCount (
   PMPrinter printer,
   UInt32* count);
```

`printer`
: A `PMPrinter` object.

`count`
: On return, the number of supported printing resolutions.

function result
: A result code. See "Result Codes" (page 144). The result code `kPMNotImplemented` indicates that the printer driver does not support multiple resolution settings.

Special Considerations

Valid within the context of a printing session.

Carbon Porting Notes

`PMPrinterGetPrinterResolutionCount` is equivalent to calling the Classic Printing Manager function `PrGeneral` with the `getRslDataOp` opcode.

PMPrinterGetPrinterResolution

Obtains the resolution setting for the specified printer according to the tag parameter.

```
OSStatus PMPrinterGetPrinterResolution(
   PMPrinter printer,
   PMTag tag,
   PMResolution* res);
```

`printer`	A `PMPrinter` object.
`tag`	Specifies the kind of resolution information required.
`res`	On return, the printer resolution setting.
function result	A result code. See "Result Codes" (page 144). The result code `kPMNotImplemented` indicates that the printer driver does not support multiple resolution settings.

Discussion

The following resolution tag constants are recognized:

- `kPMMinRange`

  The minimum resolution supported by the printer.

- `kPMMaxRange`

  The maximum resolution supported by the printer.

- `kPMMinSquareResolution`

  The minimum resolution setting for which the horizontal and vertical resolutions are equal.

- `kPMMaxSquareResolution`

The maximum resolution setting for which the horizontal and vertical resolutions are equal.

- `kPMDefaultResolution`

  The default resolution setting for the printer (typically 72 dpi).

Special Considerations

Valid within the context of a printing session.

`PMPrinterGetIndexedPrinterResolution`

Obtains a resolution setting based on an index into the range of settings supported by the specified printer.

```
OSStatus PMPrinterGetIndexedPrinterResolution (
   PMPrinter printer,
   UInt32 index,
   PMResolution* res);
```

`printer`	A `PMPrinter` object.
`index`	An index into the range of resolution settings supported by the specified printer. Index values begin at 1.
`res`	On return, the printer resolution setting.
function result	A result code. See "Result Codes" (page 144).

Discussion

You must first use the `PMPrinterGetPrinterResolutionCount` (page 64) function to obtain the number of resolution settings supported bythe specified printer.

Special Considerations

Valid within the context of a printing session.

Carbon Porting Notes

`PMPrinterGetIndexedPrinterResolution` is equivalent to calling the Classic Printing Manager function `PrGeneral` with the `getRslDataOp` opcode and obtaining a value from the `TGetRslBlk` record.

Page Format Accessor Functions

PMGetUnadjustedPaperRect

Obtains the size of the paper in points, unaffected by rotation, resolution, or scaling.

```
OSStatus PMGetUnadjustedPaperRect (
   PMPageFormat pageFormat,
   PMRect* paperSize);
```

`pageFormat`
: A `PMPageFormat` object.

`paperSize`
: On return, a rectangle describing the physical size of the paper, in points.

function result A result code. See "Result Codes" (page 144).

Special Considerations
Before using this function you must call `PMSessionValidatePageFormat` (page 39) or `PMValidatePageFormat` (page 100) to ensure that the page format object is valid.

Valid within a printing session after creating a page format object.

Version Notes
In CarbonLib 1.0 this function was named `PMGetPhysicalPaperSize`. You can still use `PMGetPhysicalPaperSize` (page 114), but the recommended name is `PMGetUnadjustedPaperRect`.

PMSetUnadjustedPaperRect

Requests a particular paper size, unaffected by rotation, resolution or scaling.

```
OSStatus PMSetUnadjustedPaperRect (
   PMPageFormat pageFormat,
   const PMRect* paperSize);
```

`pageFormat`
: A `PMPageFormat` object.

`paperSize`
: A rectangle describing the desired paper size, in points.

function result A result code. See "Result Codes" (page 144). The result code `kPMValueOutOfRange` indicates that the printer driver does not support the requested page size.

Discussion

After using this function you should always call `PMGetUnadjustedPaperRect` (page 67) to verify the paper size recorded by the printer driver.

Special Considerations

Valid within a printing session after creating a page format object.

Version Notes

In CarbonLib 1.0 this function was named `PMSetPhysicalPaperSize`. You can still use `PMSetPhysicalPaperSize` (page 114), but the recommended name is `PMSetUnadjustedPaperRect`.

PMGetUnadjustedPageRect

Obtains the size of the imageable area in points, unaffected by rotation, resolution, or scaling.

```
OSStatus PMGetUnadjustedPageRect (
   PMPageFormat pageFormat,
   PMRect* pageSize);
```

`pageFormat`

A `PMPageFormat` object.

`pageSize`

On return, the page drawing rectangle, in points.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePageFormat` (page 39) or `PMValidatePageFormat` (page 100) to ensure that the page format object is valid.

Valid within a printing session after creating a page format object.

Version Notes

In CarbonLib 1.0 this function was named `PMGetPhysicalPageSize`. You can still use `PMGetPhysicalPageSize` (page 115) but the recommended name is `PMGetUnadjustedPageRect`.

PMGetAdjustedPaperRect

Obtains the paper size, taking into account orientation, application drawing resolution, and scaling settings.

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

`pageFormat`	A `PMPageFormat` object.
`paperRect`	On return, a rectangle describing the current paper size, in points, taking into account scaling, rotation and resolution settings.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePageFormat` (page 39) or `PMValidatePageFormat` (page 100) to ensure that the page format object is valid.

Valid within a printing session after creating a page format object.

Carbon Porting Notes

`PMGetAdjustedPaperRect` is equivalent to getting the Classic print record's `rPaper` field.

PMGetAdjustedPageRect

Obtains the page size, taking into account orientation, application drawing resolution, and scaling settings.

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

`pageFormat`	A `PMPageFormat` object.
`pageRect`	On return, a rectangle describing the current page size, in points, taking into account scaling, rotation, and resolution settings.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePageFormat` (page 39) or `PMValidatePageFormat` (page 100) to ensure that the page format object is valid.

Valid within a printing session after creating a page format object.

Carbon Porting Notes

`PMGetAdjustedPageRect` is equivalent to getting the Classic print record's `TPrInfo.rPage` field.

PMSetAdjustedPageRect

Requests a particular page size, independent of the current rotation, resolution, or scaling settings.

```
OSStatus PMSetAdjustedPageRect (
   PMPageFormat pageFormat,
   const PMRect* paperSize);
```

`pageFormat`
: A `PMPageFormat` object.

`paperSize`
: The desired page size, in points.

function result A result code. See "Result Codes" (page 144).

Discussion

You can use `PMSetAdjustedPageRect` to create a drawing rectangle without going through the Page Setup dialog box or calling other page format accessor functions. This capability is intended for use by sophisticated publishing applications for which the Page Setup dialog box does not provide sufficient control. Typically such applications display their own specialized document format dialog box. This function allows an application to specify the dimensions of the pixel map into which it will draw.

Special Considerations

Valid within a printing session after creating a page format object.

PMGetScale

Obtains the scaling factor currently applied to the page and paper rectangles.

```
OSStatus PMGetScale (
   PMPageFormat pageFormat,
   double* scale);
```

`pageFormat`	A `PMPageFormat` object.
`scale`	On return, a pointer to a variable containing the scaling factor expressed as a percentage. For example, a value of 100 means 100 percent (that is, no scaling); a value of 50 means 50 percent scaling.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePageFormat` (page 39) or `PMValidatePageFormat` (page 100) to ensure that the page format object is valid.

Valid within a printing session after creating a page format object.

PMSetScale

Sets the scaling factor for the page and paper rectangles.

```
OSStatus PMSetScale (
   PMPageFormat pageFormat,
   double scale);
```

`pageFormat`	A `PMPageFormat` object.
`scale`	The desired scaling factor expressed as a percentage. For example, for 50 percent scaling, pass a value of 50; for no scaling, pass 100.
function result	A result code. See "Result Codes" (page 144).

Discussion

Use `PMSetScale` to change the scaling factor that will appear when your application invokes the Page Setup dialog box.

Special Considerations

Valid within a printing session after creating a page format object.

PMGetResolution

Obtains the current application drawing resolution.

```
OSStatus PMGetResolution (
   PMPageFormat pageFormat,
   PMResolution* res);
```

`pageFormat`	A `PMPageFormat` object.
`res`	On return, the application drawing resolution.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePageFormat` (page 39) or `PMValidatePageFormat` (page 100) to ensure that the page format object is valid.

Valid within a printing session after creating a page format object.

Carbon Porting Notes

`PMGetResolution` is equivalent to accessing the `iVRes` and `iHRes` fields from Classic print record's `TPrInfo` record.

PMSetResolution

Sets the application drawing resolution.

```
OSStatus PMSetResolution (
   PMPageFormat pageFormat,
   const PMResolution* res);
```

`pageFormat`	A `PMPageFormat` object.
`res`	The desired application drawing resolution.
function result	A result code. See "Result Codes" (page 144).

Discussion

Before setting your application drawing resolution, you should determine the valid drawing resolutions supported by the current printer, using `PMPrinterGetPrinterResolutionCount` (page 64), `PMPrinterGetPrinterResolution` (page 65) and `PMPrinterGetIndexedPrinterResolution` (page 66).

Special Considerations

Valid within a printing session after creating a page format object.

PMGetOrientation

Obtains the current setting for page orientation.

```
OSStatus PMGetOrientation (
   PMPageFormat pageFormat,
   PMOrientation* orientation);
```

`pageFormat`	A `PMPageFormat` object.
`orientation`	On return, a value representing the page orientation. See "Page Orientation Constants" (page 139) for a list of possible return values.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePageFormat` (page 39) or `PMValidatePageFormat` (page 100) to ensure that the page format object is valid.

Valid within a printing session after creating a page format object.

Carbon Porting Notes

`PMGetOrientation` is equivalent to calling the Classic Printing Manager function `PrGeneral` with the `getRotnOp` opcode.

PMSetOrientation

Sets the page orientation for printing.

```
OSStatus PMSetOrientation (
   PMPageFormat pageFormat,
   PMOrientation orientation,
   Boolean lock);
```

`pageFormat`
: A `PMPageFormat` object.

`orientation`
: The desired orientation setting. See “Page Orientation Constants” (page 139) for a list of values you can use to specify page orientation.

`lock`
: Pass `true` to prevent the user from changing the orientation. Locking is supported only on Mac OS X.

function result A result code. See “Result Codes” (page 144). The result code `kPMLockIgnored` indicates that the printer driver does not support locking of the orientation value.

Special Considerations

Valid within a printing session after creating a page format object.

Carbon Porting Notes

`PMSetOrientation` is equivalent to setting the orientation bit(s) in the low byte of the Classic print record `wDev` field.

Print Settings Accessor Functions

PMGetCopies

Obtains the number of copies that the user has requested to be printed.

```
OSStatus PMGetCopies (
   PMPrintSettings printSettings,
   UInt32* copies);
```

`printSettings`
: A `PMPrintSettings` object.

`copies`
: On return, the number of copies requested by the user.

function result A result code. See “Result Codes” (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePrintSettings` (page 41) or `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid within a printing session after creating a print settings object.

Carbon Porting Notes

`PMGetCopies` is equivalent to using the Classic print record field `TPrJob.iCopies`.

PMSetCopies

Sets the default value to be displayed in the Print dialog box for the number of copies to be printed.

```
OSStatus PMSetCopies (
   PMPrintSettings printSettings,
   UInt32 copies,
   Boolean lock);
```

`printSettings`
: A `PMPrintSettings` object.

`copies`
: The requested default value for the number of copies to print.

`lock`
: Pass `true` to prevent the user from changing the number of copies. Locking is supported only on Mac OS X.

function result
: A result code. See "Result Codes" (page 144). The result code `kPMLockIgnored` indicates that the printer driver does not support locking of the `copies` value.

Special Considerations

Valid within a printing session after creating a print settings object.

Carbon Porting Notes

`PMSetCopies` is equivalent to changing the Classic print record field `TPrJob.iCopies`.

PMGetFirstPage

Obtains the number of the first page to be printed.

```
OSStatus PMGetFirstPage (
   PMPrintSettings printSettings,
   UInt32* first);
```

`printSettings`	A `PMPrintSettings` object.
`first`	On return, a pointer to a variable containing the page number of the first page to print. The default first page number is 1.
function result	A result code. See "Result Codes" (page 144).

Discussion

You use this function to obtain the page number entered by the user in the From field of the Print dialog box. If the user did not enter a value, the function returns the value of the previous call to `PMSetFirstPage` (page 76), if any, or the default value of 1.

Special Considerations

Before using this function you must call `PMSessionValidatePrintSettings` (page 41) or `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid within a printing session after creating a print settings object.

Carbon Porting Notes

`PMGetFirstPage` is equivalent to using the Classic print record's `TprJob.iFstPage` field.

PMSetFirstPage

Sets the default page number of the first page to be printed, as displayed in the Print dialog box.

```
OSStatus PMSetFirstPage (
   PMPrintSettings printSettings,
   UInt32 first,
   Boolean lock);
```

`printSettings`
: A `PMPrintSettings` object.

`first`
: The page number of the first page to print. This value appears in the From field of the Print dialog box.

`lock`
: Pass `true` to prevent the user from changing the first page value. Locking is supported only on Mac OS X.

function result A result code. See "Result Codes" (page 144). The result code `kPMLockIgnored` indicates that the printer driver does not support locking of the `first` value.

Special Considerations

Valid within a printing session after creating a print settings object.

Carbon Porting Notes

`PMSetFirstPage` is equivalent to changing the Classic print record's `TprJob.iFstPage` field.

`PMGetLastPage`

Obtains the number of the last page to be printed.

```
OSStatus PMGetLastPage (
   PMPrintSettings printSettings,
   UInt32* last);
```

`printSettings`
: A `PMPrintSettings` object.

`last`
: On return, a pointer to a variable containing the page number of the last page to print.

function result A result code. See "Result Codes" (page 144).

Discussion

You use this function to obtain the page number entered by the user in the To field of the Print dialog box. If the user did not enter a value, the function returns the value of the previous call to `PMSetLastPage` (page 78), if any, or the default value. On Mac OS 8 and 9, the default and maximum last page number is `9999`. On Mac OS X, the default and maximum last page number is `MAX_INT(UInt32)`.

Special Considerations

Before using this function you must call `PMSessionValidatePrintSettings` (page 41) or `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid within a printing session after creating a print settings object.

Carbon Porting Notes

`PMGetLastPag` corresponds to using the Classic print record's `TprJob.iLstPage` field.

PMSetLastPage

Sets the default page number of the last page to be printed, as displayed in the Print dialog box.

```
OSStatus PMSetLastPage (
   PMPrintSettings printSettings,
   UInt32 last,
   Boolean lock);
```

`printSettings`
: A `PMPrintSettings` object.

`last`
: The page number of the last page to print. This value appears in the To field of the Print dialog box. Pass the constant `kPMPrintAllPages` to print the entire document. This causes the All button to be selected, and clears the From and To fields of the Print dialog box.

`lock`
: Pass `true` to prevent the user from changing the last page value. Locking is supported only on Mac OS X.

function result A result code. See "Result Codes" (page 144). The result code `kPMLockIgnored` indicates that the printer driver does not support locking of the `last` value.

Discussion

On Mac OS 8 and 9, the default and maximum last page number is `9999`. On Mac OS X, the default and maximum last page number is `MAX_INT(UInt32)`.

Special Considerations

Valid within a printing session after creating a print settings object.

Carbon Porting Notes

`PMSetLastPage` is equivalent to changing the Classic print record's `TprJob.iLstPage` field.

PMGetPageRange

Obtains the valid range of pages that can be printed.

```
OSStatus PMGetPageRange (
   PMPrintSettings printSettings,
   UInt32* minPage,
   UInt32* maxPage);
```

`printSettings`	A `PMPrintSettings` object.
`minPage`	On return, the minimum allowable page number.
`maxPage`	On return, the maximum allowable page number.
function result	A result code. See "Result Codes" (page 144).

Discussion

The default page range is 1-32000. This range is independent of the first and last page values reported by `PMGetFirstPage` (page 76) and `PMGetLastPage` (page 77). See `PMSetPageRange` (page 80) for more information.

Special Considerations

Before using this function you must call `PMSessionValidatePrintSettings` (page 41) or `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid within a printing session after creating a print settings object.

Carbon Porting Notes

No corresponding Classic Printing Manager function. The page range returned by this function is unrelated to the information contained in the Classic print record's `TprJob.iFstPage` and `TprJob.iLstPage` fields.

PMSetPageRange

Sets the valid range of pages that can be printed.

```
OSStatus PMSetPageRange (
   PMPrintSettings printSettings,
   UInt32 minPage,
   UInt32 maxPage);
```

`printSettings`
: A `PMPrintSettings` object.

`minPage`
: The minimum allowable page number. On Mac OS X, this value will appear as the default in the From field of the Print dialog box.

`maxPage`
: The maximum allowable page number. On Mac OS X, this value will appear as the default in the To field of the Print dialog box. Pass the constant `kPMPrintAllPages` to allow the user to print the entire document. On Mac OS X, this causes the All button to be selected,and clears the From and To fields of the Print dialog box.

function result A result code. See "Result Codes" (page 144).

Discussion

This function allows applications running on Mac OS X to set the minimum and maximum page numbers that can be printed for a document. If the user enters a value outside of this range in the Print dialog box, the Carbon Printing Manager displays an alert message.

Because the Classic Printing Manager does not support this feature, the page range cannot be automatically enforced in the Print dialog box on Mac OS 8 or 9. If your application displays the Print dialog box and the user enters values outside of the range you specify with `PMSetPageRange`, the values entered by the user will take precedence. (You can use the `PMGetFirstPage` (page 76) and `PMGetLastPage` (page 77) functions to obtain the values entered by the user in the Print dialog box.)

In all cases, if your application sets a range with `PMSetPageRange` and subsequently calls `PMSetFirstPage` (page 76) or `PMSetLastPage` (page 78) with values outside of the specified range, the Carbon Printing Manager will return an error result of `kPMValueOutOfRange`. Conversely, if your application calls `PMSetPageRange` after calling `PMSetFirstPage` or `PMSetLastPage` (or after displaying the Print dialog box), the page range specified by `PMSetPageRange` will take precedence, and the first and last page values will be adjusted accordingly.

Special Considerations

Valid within a printing session after creating a print settings object.

Carbon Porting Notes

No corresponding Classic Printing Manager function. The page range specified by this function is unrelated to the information contained in the Classic print record's `TprJob.iFstPage` and `TprJob.iLstPage` fields.

PMGetColorMode

Obtains the color mode for the print job.

```
OSStatus PMGetColorMode  (
   PMPrintSettings printSettings,
   PMColorMode* colorMode);
```

`printSettings`
: A `PMPrintSettings` object.

`colorMode`
: On return, the color mode setting (black and white, grayscale, or color). See "Color Mode Constants" (page 137) for a list of possible return values.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePrintSettings` (page 41) or `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid within a printing session after creating a print settings object.

PMSetColorMode

Sets the desired color mode for the print job.

```
OSStatus PMSetColorMode (
   PMPrintSettings printSettings,
   PMColorMode colorMode);
```

`printSettings`
: A `PMPrintSettings` object.

`colorMode`
: The desired color mode (black and white, grayscale, or color). See "Color Mode Constants" (page 137) for a list of values you can use to specify the color mode.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid within a printing session after creating a print settings object.

PMGetJobNameCFString

Obtains the name of the print job.

```
OSStatus PMGetJobNameCFString (
   PMPrintSettings printSettings,
   CFStringRef* name);
```

`printSettings`
: A `PMPrintSettings` object.

`name`
: On return, the name of the print job.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Before using this function you must call `PMSessionValidatePrintSettings` (page 41) or `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid within a printing session after creating a print settings object.

PMSetJobNameCFString

Specifies the name of a print job.

```
OSStatus PMSetJobNameCFString (
   PMPrintSettings printSettings,
   CFStringRef name);
```

`printSettings`
: A `PMPrintSettings` object.

`name`
: The name to assign to the print job. This string will be used to name the spool file.

function result
: A result code. See "Result Codes" (page 144). The result code `kPMInvalidParameter` is returned if you pass `NULL` or an empty string in the `name` parameter.

Special Considerations

Valid within a printing session after creating a print settings object.

PMGetDestination

Obtains the destination of a print job.

```
OSStatus PMGetDestination (
   PMPrintSettings printSettings,
   PMDestinationType* destType,
   CFURLRef* fileURL);
```

`printSettings`
: A `PMPrintSettings` object.

`destType`
: On return, the destination for the print job (printer, file, or fax). See "DestinationType Constants" (page 137) for a list of possible return values.

`fileURL`
: On return, the location of the print job destination.

function result
: A result code. See "Result Codes" (page 144).

Discussion

For print jobs that are sent to disk, as opposed a printer, you can use this function to obtain the location of the destination file.

Special Considerations

Before using this function you must call `PMSessionValidatePrintSettings` (page 41) or `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid within a printing session after creating a print settings object.

Print Session Accessor Functions

PMSessionGetDocumentFormatGeneration

Obtains the spool file formats that can be generated by the print spooler.

```
OSStatus PMSessionGetDocumentFormatGeneration (
   PMPrintSession printSession,
   CFArrayRef* docFormats);
```

`printSession`	A `PMPrintSession` object.
`docFormats`	On return, an array of `CFString` values containing MIME types specifying the available spool file formats.
function result	A result code. See "Result Codes" (page 144).

Discussion

Spool file formats are represented by MIME types. The formats supported on Mac OS 8 and 9 are PICT and PICT+PS. The Mac OS X print spooler supports PDF, PS, PICT, and PICT+PS.

On Mac OS 8 and 9, the spool file format is determined by the current printer driver, which is PICT+PS for all PostScript drivers and PICT for all other drivers. On Mac OS X the default spool file format is PDF.

Special Considerations

Valid within the context of a printing session.

PMSessionSetDocumentFormatGeneration

Requests a specified spool file format and identifies the graphics contexts that will be used to draw pages within the print loop.

```
OSStatus PMSessionSetDocumentFormatGeneration (
   PMPrintSession printSession,
   CFStringRef docFormat,
   CFArrayRef graphicsContexts,
   CFTypeRef options);
```

`printSession`
: A `PMPrintSession` object.

`docFormat`
: Your desired spool file format, specified as a MIME type in a `CFString`. See "Document Format Strings" (page 138) for a description of the constants you can use to specify the document format.

`graphicsContexts`
: An array of graphics contexts that will be used to draw pages within the print loop.

`options`
: Reserved for future use.

function result A result code. See "Result Codes" (page 144).

Discussion

The only graphics context currently available to Carbon applications is QuickDraw (`kPMGraphicsContextQuickdraw`). Before requesting a spool file format using this function, you should call the function `PMSessionGetDocumentFormatGeneration` (page 84) to get the list of supported formats.

Special Considerations

Valid within the context of a printing session.

PMSessionGetDocumentFormatSupported

Obtains the spool file formats that are accepted by the current printer driver.

```
OSStatus PMSessionGetDocumentFormatSupported (
   PMPrintSession printSession,
   CFArrayRef* docFormats,
   UInt32 limit);
```

`printSession`
: A `PMPrintSession` object.

`docFormats`
: On return, an array of `CFString` values containing MIME types specifying the spool file formats supported by the current printer driver. See "Document Format Strings" (page 138) for a description of possible return values.

`limit`
: The maximum number of supported document formats to be returned.

function result
: A result code. See "Result Codes" (page 144).

Discussion

Spool file formats are represented by MIME types. On Mac OS 8 and 9, raster printers accept only PICT spool files, PostScript printer drivers accept PICT and PICT+PS spool files. On Mac OS X, printer modules may support a wide range of spool file formats. The first item in the list of supported spool file formats is the default for the current printer driver.

Special Considerations

Valid within the context of a printing session.

PMSessionIsDocumentFormatSupported

Reports whether the current printer driver supports a specified spool file format.

```
OSStatus PMSessionIsDocumentFormatSupported (
   PMPrintSession printSession,
   CFStringRef docFormat,
   Boolean* supported);
```

`printSession`
: A `PMPrintSession` object.

`docFormat`
: A spool file format represented by a MIME type.

`supported`
: Returns `true` if the spool file format is supported by the current printer driver.

function result
: A result code. See "Result Codes" (page 144).

Special Considerations

Valid within the context of a printing session.

PMSessionGetGraphicsContext

Obtains the current graphics context.

```
OSStatus PMSessionGetGraphicsContext (
   PMPrintSession printSession,
   CFStringRef graphicsType,
   void** graphicsContext);
```

`printSession`	A `PMPrintSession` object.
`graphicsType`	The desired graphics context type.
`graphicsContext`	On return, a reference to the current graphics context. This reference must be typecast to an appropriate structure (such as a `grafPort`) by the caller.
function result	A result code. See "Result Codes" (page 144).

Discussion

On Mac OS 8 and 9, applications are limited to a QuickDraw context, so `PMSessionGetGraphicsContext` returns the printing graphics port. On Mac OS X, the graphics context is an opaque graphics port. The `graphicsType` argument is currently ignored, but a future version of the Carbon Printing Manager will support the ability to switch between QuickDraw and Core Graphics contexts on Mac OS X.

Special Considerations

Valid within the context of a printing session after calling `PMSessionBeginDocument`.

Carbon Porting Notes

`PMSessionGetGraphicsContext` is equivalent to accessing the Classic Printing Manager's `TPrPort.gPort` field.

Printing with PostScript and ColorSync

PMSessionPostScriptBegin

Puts the current printer driver into PostScript mode, ready to accept PostScript data instead of QuickDraw data.

```
OSStatus PMSessionPostScriptBegin (
   PMPrintSession printSession);
```

`printSession`	A `PMPrintSession` object.
function result	A result code. See “Result Codes” (page 144).

Discussion
To ensure that the current printer driver supports PostScript data, call `PMSessionGetDocumentFormatSupported` or `PMSessionIsDocumentFormatSupported` before calling `PMSessionPostScriptBegin`.

Special Considerations
Valid within a `PMSessionBeginPage`/`PMSessionEndPage` block.

Carbon Porting Notes
`PMSessionPostScriptBegin` is equivalent to calling the Classic Printing Manager function `PicComment` with the `PostScriptBegin` picture comment.

PMSessionPostScriptEnd

Restores the current driver to QuickDraw mode, ready to accept QuickDraw data instead of PostScript data.

```
OSStatus PMSessionPostScriptEnd (
   PMPrintSession printSession);
```

`printSession`	A `PMPrintSession` object.
function result	A result code. See “Result Codes” (page 144).

Discussion
Call `PMSessionPostScriptEnd` to complete a PostScript session started with `PMSessionPostScriptBegin`.

Special Considerations

Valid within a `PMSessionBeginPage`/`PMSessionEndPage` block.

Carbon Porting Notes

`PMSessionPostScriptEnd` is equivalent to calling the Classic Printing Manager function `PicComment` with the `PostScriptEnd` picture comment.

PMSessionPostScriptHandle

Passes PostScript data, referenced by a handle, to the current printer driver.

```
OSStatus PMSessionPostScriptHandle (
   PMPrintSession printSession,
   Handle psHandle);
```

`printSession`	A `PMPrintSession` object.
`psHandle`	A reference to PostScript data.
function result	A result code. See "Result Codes" (page 144).

Special Considerations

Valid between calls to `PMSessionPostScriptBegin` and `PMSessionPostScriptEnd`.

Carbon Porting Notes

`PMSessionPostScriptHandle` is equivalent to calling the Classic Printing Manager function `PicComment` with the `PostScriptHandle` picture comment.

PMSessionPostScriptData

Passes PostScript data, referenced by a pointer, to the current printer driver.

```
OSStatus PMSessionPostScriptData (
   PMPrintSession printSession,
   Ptr psPtr,
   Size len);
```

`printSession`	A `PMPrintSession` object.
`psPtr`	A pointer to PostScript data.

`len`
: The number of bytes of PostScript data to pass to the current driver.

function result A result code. See “Result Codes” (page 144).

Special Considerations

Valid between calls to `PMSessionPostScriptBegin` and `PMSessionPostScriptEnd`.

PMSessionPostScriptFile

Passes PostScript data, contained in a file, to the current printer driver.

```
OSStatus PMSessionPostScriptFile (
   PMPrintSession printSession,
   FSSpec* psFile);
```

`printSession`
: A `PMPrintSession` object.

`psFile`
: A file specification.

function result A result code. See “Result Codes” (page 144).

Special Considerations

Valid between calls to `PMSessionPostScriptBegin` and `PMSessionPostScriptEnd`.

Carbon Porting Notes

`PMSessionPostScriptFile` is equivalent to calling the Classic Printing Manager function `PicComment` with the `PostScriptFile` picture comment.

PMSessionEnableColorSync

Enables ColorSync color matching for the current page.

```
OSStatus PMSessionEnableColorSync (
   PMPrintSession printSession);
```

`printSession`
: A `PMPrintSession` object.

function result A result code. See “Result Codes” (page 144).

Special Considerations

Valid between calls to `PMSessionBeginPage` and `PMSessionEndPage`.

Carbon Porting Notes

`PMSessionEnableColorSync` is equivalent to calling the Classic Printing Manager function `PicComment` with the `CMEnableMatching` picture comment.

`PMSessionDisableColorSync`

Disables ColorSync color matching for the current page.

```
OSStatus PMSessionDisableColorSync (
   PMPrintSession printSession);
```

`printSession`
A `PMPrintSession` object.

function result A result code. See "Result Codes" (page 144).

Special Considerations

Valid between calls to `PMSessionBeginPage` and `PMSessionEndPage`.

Carbon Porting Notes

`PMSessionDisableColorSync` is equivalent to calling the Classic Printing Manager function `PicComment` with the `CMDisableMatching` picture comment.

`PMSetProfile`

Embeds a color profile during printing.

```
OSStatus PMSetProfile (
   PMPrintSettings printSettings,
   PMTag tag,
   const CMProfileLocation* profile);
```

`printSettings`
A `PMPrintSettings` object.

`tag`
A tag specifying the profile. Currently, only `kPMSourceProfile` is supported.

`profile`
The location of the profile.

function result A result code. See "Result Codes" (page 144).

Discussion
Use `PMSetProfile` to take advantage of ColorSync color management. Call `PMSetProfile` each time you want to change the profile for drawing page elements.

Special Considerations
Valid within a printing session after creating a print settings object.

Carbon Porting Notes
`PMSetProfile` is equivalent to calling the Classic Printing Manager function `PicComment` with the `CMBeginProfile` picture comment.

Compatibility Functions

PMSessionError

Obtains the result code from the last Carbon Printing Manager function called in the specified printing session.

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

`printSession`	A `PMPrintSession` object.
function result	A result code. See "Result Codes" (page 144). The result code `kPMCancel` indicates the user canceled the current print job.

Discussion
You can use this function to determine whether the user has canceled the current print job. The result code `kPMCancel` (page 142) is returned if the user clicks the Cancel button in the printer driver's status dialog box. If this or any other error is encountered during the print loop, your application should call the appropriate "end" routines (for example, `PMSessionEndPage` and `PMSessionEndDocument`) to exit the print loop before reporting the error.

Special Considerations
Valid after calling `PMCreateSession`.

Carbon Porting Notes
Because all Carbon Printing Manager functions return a result code, the `PMSessionError` function is not required for general error checking, but can be used in your print loop to determine if the user canceled a print job.

PMSessionError is equivalent to the Classic Printing Manager function PrError.

PMSessionSetError

Sets the value of the current Carbon Printing Manager result code for the specified printing session.

```
OSStatus PMSessionSetError (
   PMPrintSession printSession,
   OSStatus printError);
```

printSession	A PMPrintSession object.
printError	The result code you wish to set. This result code will be returned by the PMSessionError function.
function result	A result code. See “Result Codes” (page 144).

Discussion
You can use this function to terminate a printing session if your application encounters any errors inside the print loop.

Special Considerations
Valid after calling PMCreateSession.

Carbon Porting Notes
PMSessionSetError is equivalent to the Classic Printing Manager function PrSetError.

PMSessionGeneral

Maintains compatibility with the Classic Printing Manager’s PrGeneral function.

```
OSStatus PMSessionGeneral (
   PMPrintSession printSession,
   Ptr pData);
```

printSession	A PMPrintSession object.
pData	A pointer to a PrGeneral data structure.

function result A result code. See "Result Codes" (page 144).

Discussion
You should use the page format and print settings accessor functions instead of `PMSessionGeneral`.

Special Considerations
Valid within the context of a printing session.

Carbon Porting Notes
Use of the `PMSessionGeneral` function is not required or recommended, and support for it may be limited in future versions of the Carbon Printing Manager.

`PMSessionGeneral` is equivalent to the Classic Printing Manager function `PrGeneral`.

Print Loop Functions (Non-Session)

PMBegin

Prepares the Carbon Printing Manager for use.

```
OSStatus PMBegin (void);
```

function result A result code. See "Result Codes" (page 144).

Discussion
`PMCreateSession` (page 34) is recommended instead of `PMBegin`.

Carbon Porting Notes
`PMBegin` is equivalent to the Classic Printing Manager function `PrOpen`.

PMEnd

Closes the Carbon Printing Manager and releases its allocated memory.

```
OSStatus PMEnd (void);
```

function result A result code. See "Result Codes" (page 144).

Discussion

PMRelease (page 34) with a PMPrintSession parameter is recommended instead of PMEnd.

Carbon Porting Notes

PMEnd is equivalent to the Classic Printing Manager function PrClose.

PMBeginDocument

Establishes a graphics context for imaging a document. This context is an opaque grafPort.

```
OSStatus PMBeginDocument (
   PMPrintSettings printSettings,
   PMPageFormat pageFormat,
   PMPrintContext* printContext);
```

printSettings	A PMPrintSettings object.
pageFormat	A PMPageFormat object.
printContext	On return, an initialized PMPrintContext object.
function result	A result code. See "Result Codes" (page 144).

Discussion

PMSessionBeginDocument (page 35) is recommended instead of PMBeginDocument.

Special Considerations

Valid within a PMBegin/PMEnd bock. You must balance a call to PMBeginDocument with a call to PMEndDocument.

Carbon Porting Notes

PMBeginDocument is equivalent to the Classic Printing Manager function PrOpenDoc. However, PMBeginDocument does not include a parameter for specifying an input/output buffer because drivers supported by the Carbon Printing Manager must handle their own I/O.

PMEndDocument

Closes the context created for imaging a document.

```
OSStatus PMEndDocument (PMPrintContext printContext);
```

`printContext`	On return, an invalidated `PMPrintContext` object.
function result	A result code. See "Result Codes" (page 144).

Discussion

`PMSessionEndDocument` (page 36) is recommended instead of `PMEndDocument`.

Special Considerations

Valid after calling `PMBegin` and `PMBeginDocument`.

Carbon Porting Notes

`PMEndDocument` is equivalent to the Classic Printing Manager function `PrCloseDoc`.

PMBeginPage

Initializes a scaling rectangle for printing a page.

```
OSStatus PMBeginPage (
   PMPrintContext printContext,
   const PMRect* pageFrame);
```

`printContext`	A `PMPrintContext` object.
`pageFrame`	A pointer to a bounding rectangle for drawing the page. This value is usually obtained from the `PMGetAdjustedPageRect` (page 69) function, but if no scaling is needed, pass `NULL`. On Mac OS 8 and 9, this parameter is passed to the printer driver. Classic printer drivers typically scale the contents of the page to fit the specified rectangle. This parameter is ignored when running on Mac OS X.
function result	A result code. See "Result Codes" (page 144).

Discussion

Because PMBeginPage also initializes the printing port, your application should not make assumptions about the state of the port (for example, the current font) between successive pages. After each call to PMBeginPage, your application should call PMGetGrafPtr (page 117) to obtain the current printing port, and set the drawing port to this port using the QuickDraw SetPort function.

PMSessionBeginPage (page 36) is recommended instead of PMBeginPage.

Special Considerations

Valid after calling PMBegin and PMBeginDocument.

Carbon Porting Notes

PMBeginPage is equivalent to the Classic Printing Manager function PrOpenPage.

PMEndPage

Finishes printing the current page.

```
OSStatus PMEndPage (PMPrintContext printContext);
```

printContext
A PMPrintContext object.

function result A result code. See "Result Codes" (page 144).

Discussion

PMSessionEndPage (page 37) is recommended instead of PMEndPage.

Special Considerations

Valid after calling PMBegin, PMBeginDocument and PMBeginPage.

Carbon Porting Notes

PMEndPage is equivalent to the Classic Printing Manager function PrClosePage.

PMSetIdleProc

Installs an idle callback function in your print loop.

```
OSStatus PMSetIdleProc (PMIdleUPP idleProc);
```

`idleProc`
: A universal procedure pointer to your idle function. Your idle function is defined by the callback `PMIdleProcPtr` (page 126).

function result A result code. See "Result Codes" (page 144).

Discussion

The Carbon Printing Manager calls your idle function periodically during your print loop. Note that your idle function should not be used to display printing progess information, as that is handled automatically by the Carbon Printing Manager.

`PMSessionSetIdleProc` (page 37) is recommended instead of `PMSetIdleProc`.

Special Considerations

Valid after calling `PMBegin`.

Carbon Porting Notes

`PMSetIdleProc` is equivalent to setting the `TPrJob.pIdleProc` field in the Classic print record (`TPrint`).

Creating and Using Page Format and Print Settings Objects (Non-Session)

PMNewPageFormat

Creates a new `PMPageFormat` object.

```
OSStatus PMNewPageFormat (PMPageFormat* pageFormat);
```

`pageFormat`
: On return, a initialized `PMPageFormat` object.

function result A result code. See "Result Codes" (page 144).

Discussion

`PMNewPageFormat` allocates memory for a new `PMPageFormat` object in your application's memory space. The new page format object is empty until you set its values, or until you call `PMDefaultPageFormat` (page 99) or `PMValidatePageFormat` (page 100).

`PMCreatePageFormat` (page 38) is recommended instead of `PMNewPageFormat`.

Special Considerations
Valid after calling `PMBegin`.

PMDisposePageFormat

Releases memory previously allocated for a `PMPageFormat` object.

```
OSStatus PMDisposePageFormat (PMPageFormat pageFormat);
```

`pageFormat`	On return, an invalidated `PMPageFormat` object.
function result	A result code. See "Result Codes" (page 144).

Discussion
`PMRelease` (page 34) with a `PMPageFormat` parameter is recommended instead of `PMDisposePageFormat`.

Special Considerations
Valid after calling `PMBegin` and creating a page format object.

PMDefaultPageFormat

Assigns default parameter values to an existing `PMPageFormat` object, for the current printer.

```
OSStatus PMDefaultPageFormat (PMPageFormat pageFormat);
```

`pageFormat`	On return, a `PMPageFormat` object containing default parameter values.
function result	A result code. See "Result Codes" (page 144).

Discussion
`PMSessionDefaultPageFormat` (page 39) is recommended instead of `PMDefaultPageFormat`.

Special Considerations
Valid after calling `PMBegin` and creating a page format object.

Carbon Porting Notes

`PMDefaultPageFormat` combined with `PMDefaultPrintSettings` is equivalent to the Classic Printing Manager's `PrintDefault` function.

PMValidatePageFormat

Obtains a valid `PMPageFormat` object.

```
OSStatus PMValidatePageFormat (
   PMPageFormat pageFormat,
   Boolean* result);
```

`pageFormat`	A `PMPageFormat` object to be validated.
`result`	Returns `true` if any parameters were changed, `false` if no changes were required.
function result	A result code. See "Result Codes" (page 144).

Discussion

`PMSessionValidatePageFormat` (page 39) is recommended instead of `PMValidatePageFormat`.

Special Considerations

Valid after calling `PMBegin` and creating a page format object.

Carbon Porting Notes

`PMValidatePageFormat` combined with `PMValidatePrintSettings` is equivalent to the Classic Printing Manager's `PrValidate` function.

PMNewPrintSettings

Creates a new `PMPrintSettings` object.

```
OSStatus PMNewPrintSettings (PMPrintSettings* printSettings);
```

`printSettings`	On return, an initialized `PMPrintSettings` object.
function result	A result code. See "Result Codes" (page 144).

Discussion
PMNewPrintSettings allocates memory for a new PMPrintSettings object in your application's memory space. The new print settings object is empty until you set its values, or until you call PMDefaultPrintSettings or PMValidatePrintSettings.

PMCreatePrintSettings (page 40) is recommended instead of PMNewPrintSettings.

Special Considerations
Valid after calling PMBegin.

PMDisposePrintSettings

Releases memory previously allocated for a PMPrintSettings object.

```
OSStatus PMDisposePrintSettings (
   PMPrintSettings printSettings);
```

printSettings	On return, an invalidated PMPrintSettings reference.
function result	A result code. See "Result Codes" (page 144).

Discussion
PMRelease (page 34) with a PMPrintSettings parameter is recommended instead of PMDisposePrintSettings.

Special Considerations
Valid after calling PMBegin and creating a print settings object.

PMDefaultPrintSettings

Assigns default parameter values to a PMPrintSettings object.

```
OSStatus PMDefaultPrintSettings (
   PMPrintSettings printSettings);
```

printSettings	A PMPrintSettings object. On return, the object contains default parameter values.
function result	A result code. See "Result Codes" (page 144).

Discussion

`PMSessionDefaultPrintSettings` (page 40) is recommended instead of `PMDefaultPrintSettings`.

Special Considerations

Valid after calling `PMBegin` and creating a print settings object.

Carbon Porting Notes

`PMDefaultPrintSettings` combined with `PMDefaultPageFormat` is equivalent to the Classic Printing Manager's `PrintDefault` function.

PMValidatePrintSettings

Obtains a valid `PMPrintSettings` object.

```
OSStatus PMValidatePrintSettings (
   PMPrintSettings printSettings,
   Boolean* result);
```

`printSettings`
: The `PMPrintSettings` object to be validated.

`result`
: On return, a value of `true` if any parameters were changed, or `false` if no changes were required.

function result A result code. See "Result Codes" (page 144).

Discussion

`PMSessionValidatePrintSettings` (page 41) is recommended instead of `PMValidatePrintSettings`.

Special Considerations

Valid after calling `PMBegin` and creating a print settings object.

Carbon Porting Notes

`PMValidatePrintSettings` combined with `PMValidatePageFormat` is equivalent to the Classic Printing Manager's `PrValidate` function.

Print Record Compatibility Functions (Non-Session)

PMConvertOldPrintRecord

Creates a new `PMPageFormat` object and a new `PMPrintSettings` object from an existing Classic print record.

```
OSStatus PMConvertOldPrintRecord (
   Handle printRecordHandle,
   PMPrintSettings* printSettings,
   PMPageFormat* pageFormat);
```

`printRecordHandle`
: A handle to a Classic print record.

`printSettings`
: On return, a validated `PMPrintSettings` object.

`pageFormat`
: On return, a validated `PMPageFormat` object.

function result A result code. See "Result Codes" (page 144).

Discussion
`PMSessionConvertOldPrintRecord` (page 49) is recommended instead of `PMConvertOldPrintRecord`.

Special Considerations
Valid after calling `PMBegin`.

PMMakeOldPrintRecord

Creates a Classic print record from a `PMPageFormat` and a `PMPrintSettings` object.

```
OSStatus PMMakeOldPrintRecord (
   PMPrintSettings printSettings,
   PMPageFormat pageFormat,
   Handle* printRecordHandle);
```

`printSettings`
: A `PMPrintSettings` object.

`pageFormat`
: A `PMPageFormat` object.

`printRecordHandle`
: On return, a handle to a Classic print record. Your application must dispose of this handle.

function result A result code. See "Result Codes" (page 144).

Discussion

Use `PMMakeOldPrintRecord` to create a Classic print record to store with your documents for compatibility with non-Carbon versions of your application. Note that because the page format and print settings objects contain more information than the Classic print record, some settings may be lost in conversion.

`PMSessionMakeOldPrintRecord` (page 49) is recommended instead of `PMMakeOldPrintRecord`.

Special Considerations

Valid after calling `PMBegin` and creating a page format and print settings object.

Displaying the Page Setup and Print Dialog Boxes (Non-Session)

PMPageSetupDialog

Displays the Page Setup dialog box and records the user's selections in a `PMPageFormat` object.

```
OSStatus PMPageSetupDialog (
   PMPageFormat pageFormat,
   Boolean* accepted);
```

`pageFormat`
: A `PMPageFormat` object.

`accepted`
: Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.

function result A result code. See "Result Codes" (page 144).

Discussion

When the Page Setup dialog box is displayed, it shows the current settings contained in the page format object. If the user changes these settings and clicks the OK button, the page format object is updated with the user's selections. If the user clicks the Cancel button, the page format object is returned unchanged.

`PMSessionPageSetupDialog` (page 50) is recommended instead of `PMPageSetupDialog`.

Special Considerations

Valid after calling `PMBegin` and creating a page format object. Never call `PMPageSetupDialog` between the pages of a document.

Carbon Porting Notes

`PMPageSetupDialog` is equivalent to the Classic Printing Manager function `PrStlDialog`.

PMPrintDialog

Displays the Print dialog box and records the user's selections in a `PMPrintSettings` object.

```
OSStatus PMPrintDialog (
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   Boolean* accepted);
```

`printSettings`	A `PMPrintSettings` object.
`pageFormat`	A `PMPageFormat` object.
`accepted`	Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.
function result	A result code. See "Result Codes" (page 144).

Discussion

When the Print dialog box is displayed, it shows the current settings in the print settings object. If the user changes these settings and clicks the OK button, the print settings object is updated with the user's selections. If the user clicks the Cancel button, the print settings object is returned unchanged.

`PMSessionPrintDialog` (page 51) is recommended instead of `PMPrintDialog`.

Special Considerations

Valid after calling `PMBegin` and creating a page format and print settings object.

Carbon Porting Notes

`PMPrintDialog` is equivalent to the Classic Printing Manager function `PrJobDialog`.

Customizing the Page Setup and Print Dialog Boxes (Non-Session)

PMPageSetupDialogInit

Initializes a custom Page Setup dialog box.

```
OSStatus PMPageSetupDialogInit (
   PMPageFormat pageFormat,
   PMDialog* newDialog);
```

`pageFormat`
: A `PMPageFormat` object.

`newDialog`
: On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMPageSetupDialogMain` (page 107) function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPageFormat` object. See the discussion of the `PMPageSetupDialogInitProcPtr` (page 127) callback function for more information.

function result A result code. See "Result Codes" (page 144).

Discussion

`PMSessionPageSetupDialogInit` (page 52) is recommended instead of `PMPageSetupDialogInit`.

Special Considerations

Valid after calling `PMBegin` and creating a page format object.

Carbon Porting Notes

`PMPageSetupDialogInit` is equivalent to the Classic Printing Manager function `PrStlInit`.

PMPageSetupDialogMain

Displays your application's customized Page Setup dialog box.

```
OSStatus PMPageSetupDialogMain (
   PMPageFormat pageFormat,
   Boolean* accepted,
   PMPageSetupDialogInitUPP myInitProc);
```

`pageFormat`	A `PMPageFormat` object.
`accepted`	Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.
`myInitProc`	A universal procedure pointer to your dialog initialization function. Your initialization function is defined by the callback `PMPageSetupDialogInitProcPtr` (page 127).
function result	A result code. See "Result Codes" (page 144).

Discussion

`PMSessionPageSetupDialogMain` (page 53) is recommended instead of `PMPageSetupDialogMain`.

Special Considerations

Valid after calling `PMBegin` and creating a page format object.

Carbon Porting Notes

`PMPageSetupDialogMain` is equivalent to the Classic Printing Manager function `PrDlgMain`.

PMPrintDialogInit

Initializes a custom Print dialog box.

```
OSStatus PMPrintDialogInit (
   PMPrintSettings printSettings,
   PMDialog* newDialog);
```

`printSettings`	A `PMPrintSettings` object.

`newDialog`
: On return, a pointer to an initialized `PMDialog` object, ready for customization by your application.

function result A result code. See "Result Codes" (page 144).

Discussion

This function is no longer recommeded. You should instead use `PMSessionPrintDialogInit` (page 54) or `PMPrintDialogInitWithPageFormat` (page 108), both of which pass page format information to the dialog object.

Special Considerations

Valid after calling `PMBegin` and creating a page format and print settings object.

Carbon Porting Notes

You should instead use `PMSessionPrintDialogInit` (page 54) or `PMPrintDialogInitWithPageFormat` (page 108) because `PMPrintDialogInit` does not pass page format information to the dialog object.

PMPrintDialogInitWithPageFormat

Initializes a custom Print dialog box.

```
OSStatus PMPrintDialogInitWithPageFormat (
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   PMDialog* newDialog);
```

`printSettings`
: A `PMPrintSettings` object.

`constPageFormat`
: A `PMPageFormat` object.

`newDialog`
: On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMPrintDialogMain` (page 109) function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPrintSettings` object. See the discussion of the `PMPrintDialogInitProcPtr` (page 129) callback function for more information.

function result A result code. See "Result Codes" (page 144).

Discussion

`PMSessionPrintDialogInit` (page 54) is recommended instead of `PMPrintDialogInitWithPageFormat`.

Special Considerations

Valid after calling `PMBegin` and creating a page format and print settings object.

Carbon Porting Notes

`PMPrintDialogInitWithPageFormat` is equivalent to the Classic Printing Manager function `PrJobInit`.

PMPrintDialogMain

Displays your application's custom Print dialog box.

```
OSStatus PMPrintDialogMain (
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   Boolean* accepted,
   PMPrintDialogInitUPP myInitProc);
```

`printSettings`	A `PMPrintSettings` object.
`constPageFormat`	A `PMPageFormat` object.
`accepted`	Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.
`myInitProc`	A universal procedure pointer to your dialog initialization function. Your initialization function is defined by the callback `PMPrintDialogInitProcPtr` (page 129).
function result	A result code. See "Result Codes" (page 144).

Discussion

`PMSessionPrintDialogMain` (page 55) is recommended instead of `PMPrintDialogMain`.

Special Considerations

Valid after calling `PMBegin` and creating a page format and print settings object.

Carbon Porting Notes

`PMPrintDialogMain` is equivalent to the Classic Printing Manager function `PrDlgMain`.

Getting Information About the Current Printer (Non-Session)

PMGetLanguageInfo

Obtains information about the current printer's imaging language.

```
OSStatus PMGetLanguageInfo (PMLanguageInfo* info);
```

`info`
: On return, a pointer to a data structure containing the printer's language level, version and release. The format of the returned data is based on the PostScript language.

function result A result code. See "Result Codes" (page 144).

Discussion

`PMGetLanguageInfo` is useful for PostScript printers but may be irrelevant for other types of printers.

`PMPrinterGetLanguageInfo` (page 62) is recommended instead of `PMGetLanguageInfo`.

Special Considerations

Valid after calling `PMBegin`.

PMGetDriverCreator

Obtains the creator of the driver associated with the current printer.

```
OSStatus PMGetDriverCreator (OSType* creator);
```

`creator`
: On return, the 4-byte creator type of the driver (for example, `'APPL'` for an Apple printer driver).

function result A result code. See "Result Codes" (page 144).

Discussion

`PMPrinterGetDriverCreator` (page 63) is recommended instead of `PMGetDriverCreator`.

Special Considerations
Valid after calling `PMBegin`.

Carbon Porting Notes
For Classic printer drivers that support the `PrGeneral` opcode `kPrinterDirectOpCode`, `PMGetDriverCreator` provides functionality equivalent to calling `PrGeneral`, passing in the `kPDVerifySelect` selector to obtain the driver's creator type and version.

`PMGetDriverReleaseInfo`

Obtains release information for the driver associated with the current printer.

```
OSStatus PMGetDriverReleaseInfo (VersRec* release);
```

`release`	On return, a pointer to a `VersRec` data structure containing the driver's short and long version strings and country code.
function result	A result code. See "Result Codes" (page 144).

Discussion
`PMPrinterGetDriverReleaseInfo` (page 64) is recommended instead of `PMGetDriverReleaseInfo`.

Special Considerations
Valid after calling `PMBegin`.

Carbon Porting Notes
`PMGetDriverReleaseInfo` is equivalent to checking the Classic print record's `iPrVersion` field.

For Classic printer drivers that support the `PrGeneral` opcode `kPrinterDirectOpCode`, `PMGetDriverReleaseInfo` provides functionality equivalent to calling `PrGeneral`, passing in the `kPDVerifySelect` selector to obtain the driver's creator type and version.

`PMGetPrinterResolutionCount`

Obtains the number of resolution settings supported by the current printer.

```
OSStatus PMGetPrinterResolutionCount (UInt32* count);
```

`count`
: On return, the number of supported printing resolutions.

function result A result code. See "Result Codes" (page 144). The result code `kPMNotImplemented` indicates that the printer driver does not support multiple resolution settings.

Discussion

`PMPrinterGetPrinterResolutionCount` (page 64) is recommended instead of `PMGetPrinterResolutionCount`.

Special Considerations

Valid after calling `PMBegin`.

Carbon Porting Notes

`PMGetPrinterResolutionCount` is equivalent to calling the Classic Printing Manager function `PrGeneral` with the `getRslDataOp` opcode.

PMGetPrinterResolution

Obtains the resolution setting for the current printer according to the tag parameter.

```
OSStatus PMGetPrinterResolution (
   PMTag tag,
   PMResolution* res);
```

`tag`
: Specifies the kind of resolution information required.

`res`
: The printer resolution setting.

function result A result code. See "Result Codes" (page 144). The result code `kPMNotImplemented` indicates that the printer driver does not support multiple resolution settings.

Discussion

The following resolution tag constants are recognized:

- `kPMMinRange`

  The minimum resolution supported by the printer.
- `kPMMaxRange`

  The maximum resolution supported by the printer.

- `kPMMinSquareResolution`

  The minimum resolution setting for which the horizontal and vertical resolutions are equal.

- `kPMMaxSquareResolution`

  The maximum resolution setting for which the horizontal and vertical resolutions are equal.

- `kPMDefaultResolution`

  The default resolution setting for the printer (typically 72 dpi).

Special Considerations

Valid after calling `PMBegin`.

Carbon Porting Notes

`PMPrinterGetPrinterResolution` is recommended instead of `PMGetPrinterResolution`.

PMGetIndexedPrinterResolution

Obtains a resolution setting based on an index into the range of settings supported by the current printer.

```
OSStatus PMGetIndexedPrinterResolution(
   UInt32 index,
   PMResolution* res);
```

`index`

An index into the range of resolution settings supported by the specified printer. Index values begin at 1.

`res`

On return, the printer resolution setting.

function result A result code. See "Result Codes" (page 144).

Discussion

You must first use the `PMGetPrinterResolutionCount` (page 111) function to obtain the number of resolution settings supported bythe current printer.

`PMPrinterGetIndexedPrinterResolution` (page 66) is recommended instead of `PMGetIndexedPrinterResolution`.

Special Considerations
Valid after calling `PMBegin`.

Carbon Porting Notes
`PMGetIndexedPrinterResolution` is equivalent to calling the Classic Printing Manager function `PrGeneral` with the `getRslDataOp` opcode and indexing into the `TGetRslBlk` record.

Page Format Accessor Functions (Non-Session)

PMGetPhysicalPaperSize

Obtains the size of the paper in points, unaffected by rotation, resolution, or scaling.

```
OSStatus PMGetPhysicalPaperSize (
   PMPageFormat pageFormat,
   PMRect* paperSize);
```

Discussion
You should instead use `PMGetUnadjustedPaperRect` (page 67).

Carbon Porting Notes
The preferred name of this function has been changed to `PMGetUnadjustedPaperRect` (page 67).

PMSetPhysicalPaperSize

Requests a particular paper size, unaffected by rotation, resolution or scaling.

```
OSStatus PMSetPhysicalPaperSize (
   PMPageFormat pageFormat,
   const PMRect* paperSize);
```

Discussion
You should instead use `PMSetUnadjustedPaperRect` (page 67).

Carbon Porting Notes
The preferred name of this function has been changed to `PMSetUnadjustedPaperRect` (page 67).

PMGetPhysicalPageSize

Obtains the size of the imageable area in points, unaffected by rotation, resolution, or scaling.

```
OSStatus PMGetPhysicalPageSize (
   PMPageFormat pageFormat,
   PMRect* pageSize);
```

Discussion
You should instead use `PMGetUnadjustedPageRect` (page 68).

Carbon Porting Notes
The preferred name of this function has been changed to `PMGetUnadjustedPageRect` (page 68).

Print Settings Accessor Functions (Non-Session)

PMGetJobName

Obtains the name of the print job.

```
OSStatus PMGetJobName (
   PMPrintSettings printSettings,
   StringPtr name);
```

`printSettings`	A `PMPrintSettings` object.
`name`	On return, the name of the print job.
function result	A result code. See “Result Codes” (page 144).

Discussion
`PMGetJobNameCFString` (page 82) is recommended instead of `PMGetJobName`.

Special Considerations
Before using this function you must call `PMValidatePrintSettings` (page 102) to ensure that the print settings object is valid.

Valid after calling `PMBegin` and creating a print settings object.

PMSetJobName

Specifies the name of the print job.

```
OSStatus PMSetJobName (
   PMPrintSettings printSettings,
   StringPtr name);
```

`printSettings`
: A `PMPrintSettings` object.

`name`
: The name to assign to the print job. This string will be used to name the spool file.

function result A result code. See "Result Codes" (page 144). The result code `kPMInvalidParameter` is returned if you attempt to set the job name to an invalid file name or a null string.

Discussion
`PMSetJobNameCFString` (page 82) is recommended instead of `PMSetJobName`.

Special Considerations
Valid after calling `PMBegin` and creating a print settings object.

Print Session Accessor Functions (Non-Session)

PMIsPostScriptDriver

Reports whether the current printer driver supports the PostScript language.

```
OSStatus PMIsPostScriptDriver (Boolean* isPostScript);
```

`isPostScript`
: Returns `true` if the current printer driver supports PostScript.

function result A result code. See "Result Codes" (page 144).

Discussion
On Mac OS 8 and 9, `PMIsPostScriptDriver` reports whether the current printer driver supports the PostScript language. On Mac OS X, `PMIsPostScriptDriver` always returns `false`.

`PMSessionIsDocumentFormatSupported` (page 86) is recommended instead of `PMIsPostScriptDriver`.

Special Considerations

Valid after calling `PMBegin`.

Carbon Porting Notes

`PMIsPostScriptDriver` is equivalent to testing the high byte of the Classic print record's `TPrStl.wDev` field (3 = LaserWriter driver).

PMGetGrafPtr

Obtains the printing port from an opaque printing context.

```
OSStatus PMGetGrafPtr (
   PMPrintContext printContext,
   GrafPtr* grafPort);
```

`printContext`	A `PMPrintContext` object.
`grafPort`	On return, a pointer to a `grafPort` defining the current printing port.
function result	A result code. See "Result Codes" (page 144).

Discussion

`PMSessionGetGraphicsContext` (page 87) is recommended instead of `PMGetGrafPtr`.

Special Considerations

Valid after calling `PMBegin` and creating a printing context.

Carbon Porting Notes

`PMGetGrafPtr` is equivalent to accessing the Classic Printing Manager's `TPrPort.gPort` field.

Printing with PostScript and ColorSync (Non-Session)

PMPostScriptBegin

Puts the current driver into PostScript mode, ready to accept PostScript data instead of QuickDraw data.

```
OSStatus PMPostScriptBegin (void);
```

function result A result code. See "Result Codes" (page 144).

Discussion

Call `PMIsPostScriptDriver` before calling `PMPostScriptBegin` to ensure that the current driver supports PostScript data.

`PMSessionPostScriptBegin` (page 88) is recommended instead of `PMPostScriptBegin`.

Special Considerations

Valid within a `PMBeginPage`/`PMEndPage` block.

Carbon Porting Notes

`PMPostScriptBegin` is equivalent to calling the Classic Printing Manager function `PicComment` with the `PostScriptBegin` picture comment.

PMPostScriptEnd

Restores the current driver to QuickDraw mode, ready to accept QuickDraw data instead of PostScript data.

```
OSStatus PMPostScriptEnd (void);
```

function result A result code. See "Result Codes" (page 144).

Discussion

Call `PMPostScriptEnd` to complete a PostScript session started with `PMPostScriptBegin`.

`PMSessionPostScriptEnd` (page 88)is recommended instead of `PMPostScriptEnd`.

Special Considerations
Valid within a `PMBeginPage`/`PMEndPage` block.

Carbon Porting Notes
`PMPostScriptEnd` corresponds to calling the Classic Printing Manager function `PicComment` with the `PostScriptEnd` picture comment.

PMPostScriptHandle

Passes PostScript data, referenced by a handle, to the current printer driver.

```
OSStatus PMPostScriptHandle (Handle psHandle);
```

`psHandle`	A reference to PostScript data.
function result	A result code. See "Result Codes" (page 144).

Discussion
`PMSessionPostScriptHandle` (page 89) is recommended instead of `PMPostScriptHandle`.

Special Considerations
Call `PMPostScriptHandle` after calling `PMPostScriptBegin` and before calling `PMPostScriptEnd`.

Carbon Porting Notes
`PMPostScriptHandle` is equivalent to calling the Classic Printing Manager function `PicComment` with the `PostScriptHandle` picture comment.

PMPostScriptData

Passes PostScript data, referenced by a pointer, to the current printer driver.

```
OSStatus PMPostScriptData (Ptr psPtr, Size len);
```

`psPtr`	A pointer to PostScript data.
`len`	The number of bytes of PostScript data to pass to the current driver.
function result	A result code. See "Result Codes" (page 144).

Discussion
`PMSessionPostScriptData` (page 89) is recommended instead of `PMPostScriptData`.

Special Considerations
Call `PMPostScriptData` after calling `PMPostScriptBegin` and before calling `PMPostScriptEnd`.

Carbon Porting Notes
No corresponding Classic Printing Manager function.

PMPostScriptFile

Passes PostScript data, contained in a file, to the current printer driver.

```
OSStatus PMPostScriptFile (FSSpec* psFile);
```

`psFile`	A file specification.
function result	A result code. See "Result Codes" (page 144).

Discussion
`PMSessionPostScriptFile` (page 90) is recommended instead of `PMPostScriptFile`.

Special Considerations
Call `PMPostScriptFile` after calling `PMPostScriptBegin` and before calling `PMPostScriptEnd`.

Carbon Porting Notes
`PMPostScriptFile` is equivalent to calling the Classic Printing Manager function `PicComment` with the `PostScriptFile` picture comment.

PMEnableColorSync

Enables ColorSync color matching for the current page.

```
OSStatus PMEnableColorSync (void);
```

function result	A result code. See "Result Codes" (page 144).

Discussion
`PMSessionEnableColorSync` (page 90) is recommended instead of `PMEnableColorSync`.

Special Considerations
Valid between calls to `PMBeginPage` and `PMEndPage`.

PMDisableColorSync

Disables ColorSync color matching for the current page.

```
OSStatus PMDisableColorSync (void);
```

function result A result code. See "Result Codes" (page 144).

Discussion
`PMSessionDisableColorSync` (page 91) is recommended instead of `PMDisableColorSync`.

Special Considerations
Valid between calls to `PMBeginPage` and `PMEndPage`.

Compatibility Functions (Non-Session)

PMError

Obtains the result code from the last Carbon Printing Manager function called by your application.

```
OSStatus PMError (void);
```

function result A result code. See "Result Codes" (page 144). The result code `kPMCancel` indicates the user canceled the current print job.

Discussion
You can use this function to determine whether the user has canceled the current print job. The result code `kPMCancel` is returned if the user clicks the Cancel button in the printer driver's status dialog box. If this or any other error is encountered during the print loop, your application should call the appropriate "end" routines (for example, `PMEndPage` and `PMEndDocument`) to exit the print loop before reporting the error.

`PMSessionError` (page 92) is recommended instead of `PMError`.

Special Considerations
Valid after calling `PMBegin`.

Carbon Porting Notes
Because all Carbon Printing Manager functions return a result code, the `PMError` function is not required for general error checking, but can be used in your print loop to determine if the user canceled a print job.

`PMError` is equivalent to the Classic Printing Manager function `PrError`.

PMSetError

Sets the value of the current Carbon Printing Manager result code.

```
OSStatus PMSetError (OSStatus printError);
```

`printError`
The result code you wish to set. This result code will be returned by the `PMError` function.

function result A result code. See "Result Codes" (page 144).

Discussion
You can use this function to terminate a printing session if your application encounters any errors inside the print loop.

`PMSessionSetError` is recommended instead of `PMSetError`.

Special Considerations
Valid after calling `PMBegin`.

Carbon Porting Notes
`PMSetError` is equivalent to the Classic Printing Manager function `PrSetError`.

PMGeneral

Maintains compatibility with the Classic Printing Manager's `PrGeneral` function.

```
OSStatus PMGeneral (Ptr pData);
```

`pData`
A pointer to a `PrGeneral` data structure.

function result A result code. See "Result Codes" (page 144).

Discussion
You should use the page format and print settings accessor functions instead of `PMGeneral`.

`PMSessionGeneral` is recommended instead of `PMGeneral`.

Special Considerations
Valid after calling `PMBegin`.

Carbon Porting Notes
`PMGeneral` is equivalent to the Classic Printing Manager function `PrGeneral`.

Creating, Calling, and Deleting Universal Procedure Pointers

NewPMIdleUPP

Creates a new universal procedure pointer to an idle callback function.

```
PMIdleUPP NewPMIdleUPP (PMIdleProcPtr userRoutine);
```

Discussion
See the `PMIdleProcPtr` (page 126) callback function.

InvokePMIdleUPP

Calls an idle callback function.

```
void InvokePMIdleUPP (PMIdleUPP userUPP);
```

Discussion
See the `PMIdleProcPtr` (page 126) callback function.

DisposePMIdleUPP

Disposes of a universal procedure pointer to an idle callback function.

```
void DisposePMIdleUPP (PMIdleUPP userUPP);
```

Discussion
See the `PMIdleProcPtr` (page 126) callback function.

NewPMPageSetupDialogInitUPP

Creates a new universal procedure pointer to Page Setup dialog initialization callback function.

```
PMPageSetupDialogInitUPP NewPMPageSetupDialogInitUPP
                             (PMPageSetupDialogInitProcPtr userRoutine);
```

Discussion
See the `PMPageSetupDialogInitProcPtr` (page 127) callback function.

InvokePMPageSetupDialogInitUPP

Calls a Page Setup dialog initialization callback function.

```
void InvokePMPageSetupDialogInitUPP (PMPageFormat pageFormat,
                                          PMDialog* theDialog,
                                          PMPageSetupDialogInitUPP userUPP);
```

Discussion
See the `PMPageSetupDialogInitProcPtr` (page 127) callback function.

DisposePMPageSetupDialogInitUPP

Disposes of a universal procedure pointer to a Page Setup dialog initialization callback function.

```
void DisposePMPageSetupDialogInitUPP (PMPageSetupDialogInitUPP userUPP);
```

Discussion
See the `PMPageSetupDialogInitProcPtr` (page 127) callback function.

NewPMPrintDialogInitUPP

Creates a new universal procedure pointer to Print dialog initialization callback function.

```
PMPrintDialogInitUPP NewPMPrintDialogInitUPP
                             (PMPrintDialogInitProcPtr userRoutine);
```

Discussion
See the `PMPrintDialogInitProcPtr` (page 129) callback function.

InvokePMPrintDialogInitUPP

Calls a Print dialog initialization callback function.

```
void InvokePMPrintDialogInitUPP (PMPrintSettings printSettings,
                                    PMDialog* theDialog,
                                    PMPrintDialogInitUPP userUPP);
```

Discussion
See the `PMPrintDialogInitProcPtr` (page 129) callback function.

DisposePMPrintDialogInitUPP

Disposes of a universal procedure pointer to a Print dialog initialization callback function.

```
void DisposePMPrintDialogInitUPP (PMPrintDialogInitUPP userUPP);
```

Discussion
See the `PMPrintDialogInitProcPtr` (page 129) callback function.

NewPMItemUPP

Creates a new universal procedure pointer to a dialog item event handling callback function.

```
PMItemUPP NewPMItemUPP (PMItemProcPtr userRoutine);
```

Discussion
See the `PMItemProcPtr` (page 130) callback function.

InvokePMItemUPP

Calls a dialog item event handling callback function.

```
void InvokePMItemUPP (DialogPtr theDialog,
                         short item,
                         PMItemUPP userUPP);
```

Discussion
See the `PMPrintDialogInitProcPtr` (page 129) callback function.

DisposePMItemUPP

Disposes of a universal procedure pointer to a dialog item event handling callback function.

```
void DisposePMItemUPP (PMItemUPP userUPP);
```

Discussion
See the `PMPrintDialogInitProcPtr` (page 129) callback function.

Callbacks

PMIdleProcPtr

Defines a pointer to an idle function the Carbon Printing Manager calls during your print loop. Your idle function can display status information during printing.

```
typedef CALLBACK_API ( void , PMIdleProcPtr ) ( void );
```

You would declare your idle function like this if you were to name it `MyPrintIdleCallback`:

```
void MyPrintIdleCallback (void);
```

Discussion
If you install an idle function using the `PMSessionSetIdleProc` (page 37) function or the `PMSetIdleProc` (page 97) function, the Carbon Printing Manager will call your idle function periodically during your print loop. Your idle function can display application status while printing, but it should not duplicate information displayed by the Carbon Printing Manager or the printer driver.

Your idle function must check whether the user has pressed Command-period, in which case your application should stop its printing operation. If your status dialog box contains a button to cancel the printing operation, your idle function should also check for clicks in the button and respond accordingly.

To provide a pointer to your idle function, you create a universal procedure pointer (UPP) of type `PMIdleUPP` (page 136), using the function `NewPMIdleUPP` (page 123). You can do so with code like the following:

```
PMIdleUPP MyPrintIdleUPP;
MyPrintIdleUPP = NewPMIdleUPP ( &MyPrintIdleCallback );
```

If you wish to call your idle function directly, you can use the `InvokePMIdleUPP` (page 123) function.

When your print job is completed, you should use the `DisposePMIdleUPP` (page 123) function to dispose of the universal procedure pointer associated with your idle function. However, if you will use the same idle function in subsequent print jobs, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Special Considerations

Valid within the context of a printing session.

Carbon Porting Notes

Your idle function is equivalent to the Classic Printing Manager callback function `MyDoPrintIdle`.

PMPageSetupDialogInitProcPtr

Defines a pointer to a dialog initialization function the Carbon Printing Manager calls before displaying your custom Page Setup dialog box. Your initialization function can append items to the Page Setup dialog box.

```
typedef CALLBACK_API ( void , PMPageSetupDialogInitProcPtr )
                        ( PMPageFormat pageFormat , PMDialog *theDialog );
```

You would declare your dialog initialization function like this if you were to name it `MyPageSetupDialogInitCallback`:

```
void MyPageSetupDialogInitCallback (
   PMPageFormat pageFormat,
   PMDialog *theDialog);
```

`pageFormat`
A `PMPageFormat` object.

`theDialog`
A pointer to the `PMDialog` object representing your custom Page Setup dialog box. You obtain this object from the `PMSessionPageSetupDialogInit` (page 52) function. Because the `PMSessionPageSetupDialogMain` (page 53) function does not pass this object to the Carbon Printing Manager, your application should store a pointer to the dialog object in a global variable or as extended data in the `PMPageFormat` object, where it is accessible by your initialization function.

Discussion

You pass a pointer to your dialog initialization function as a parameter to the `PMSessionPageSetupDialogMain` (page 53) function. The Carbon Printing Manager then calls your dialog initialization function before displaying your custom Page Setup dialog box. Your initialization function can append items to the Page Setup dialog box, and should install an event filter using the `PMSetModalFilterProc` (page 56) function.

To provide a pointer to your dialog initialization function, you create a universal procedure pointer (UPP) of type `PMPageSetupDialogInitUPP` (page 136), using the function `NewPMPageSetupDialogInitUPP` (page 124) . You can do so with code like the following:

```
PMPageSetupDialogInitUPP MyPageSetupDialogInitCallbackUPP;
MyPageSetupDialogInitCallbackUPP =
    NewPMPageSetupDialogInitUPP ( &MyPageSetupDialogInitCallback );
```

If you wish to call your dialog initialization function directly, you can use the `InvokePMPageSetupDialogInitUPP` (page 124) function.

After you are finished with your dialog initialization function, you can dispose of the UPP with the `DisposePMPageSetupDialogInitUPP` (page 124) function. However, if you will use the same dialog initialization function for subsequent print jobs, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Special Considerations

Valid within the context of a printing session after calling `PMSessionPageSetupDialogInit`.

Carbon Porting Notes

`PMPageSetupDialogInitProcPtr` is equivalent to the Classic Printing Manager callback function `MyPrDialogAppend`.

PMPrintDialogInitProcPtr

Defines a pointer to a dialog initialization function the Carbon Printing Manager calls before displaying your custom Print dialog box. Your initialization function can append items to the Print dialog box.

```
typedef CALLBACK_API ( void , PMPrintDialogInitProcPtr )
                        ( PMPrintSettings printSettings ,
                          PMDialog *theDialog );
```

You would declare your dialog initialization function like this if you were to name it `MyPrintDialogInitCallback`:

```
void MyPrintDialogInitCallback (
   PMPrintSettings printSettings,
   PMDialog *theDialog);
```

`printSettings`
: A `PMPrintSettings` object.

`theDialog`
: A pointer to the `PMDialog` object representing your custom Print dialog box. You obtain this object from the `PMSessionPrintDialogInit` (page 54) function. Because the `PMSessionPrintDialogMain` (page 55) function does not pass this object to the Carbon Printing Manager, your application should store a pointer to the dialog object in a global variable or as extended data in the `PMPrintSettings` object, where it is accessible by your initialization function.

Discussion

You pass a pointer to your dialog initialization function as a parameter to the `PMSessionPrintDialogMain` (page 55) function. The Carbon Printing Manager then calls your dialog initialization function before displaying your custom Print dialog box. Your initialization function can append items to the Print dialog box, and should install an event filter using the `PMSetModalFilterProc` (page 56) function.

To provide a pointer to your dialog initialization function, you create a universal procedure pointer (UPP) of type `PMPrintDialogInitUPP` (page 136), using the function `NewPMPrintDialogInitUPP` (page 124) . You can do so with code like the following:

```
PMPrintDialogInitUPP MyPrintDialogInitCallbackUPP;
```

```
MyPrintDialogInitCallbackUPP =
    NewPMPrintDialogInitUPP ( &MyPrintDialogInitCallback );
```

If you wish to call your dialog initialization function directly, you can use the `InvokePMPrintDialogInitUPP` (page 125) function.

After you are finished with your dialog initialization function, you can dispose of the UPP with the `DisposePMPrintDialogInitUPP` (page 125) function. However, if you will use the same dialog initialization function for subsequent print jobs, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Special Considerations

Valid within the context of a printing session after calling `PMSessionPrintDialogInit` (page 54).

Carbon Porting Notes

`PMPrintDialogInitProcPtr` is equivalent to the Classic Printing Manager callback function `MyPrDialogAppend`.

PMItemProcPtr

Defines a pointer to an dialog item event handling function. Your dialog item event handling function receives events for items you've added to your custom Page Setup or Print dialog box.

```
typedef CALLBACK_API ( void , PMItemProcPtr )
                        ( DialogPtr theDialog , short item );
```

You would declare your event handling function like this if you were to name it `MyPMItemCallback`:

```
void MyPMItemCallback (DialogPtr theDialog, short item);
```

`theDialog`
: A `PMDialog` object representing your customized Page Setup or Print dialog box.

`item`
: The number of the dialog item receiving the event.

Discussion

You must provide an event handling callback function if you add items to the Page Setup or Print dialog boxes. Use the `PMSetItemProc` (page 57) function to install your callback function.

To provide a pointer to your event handling function, you create a universal procedure pointer (UPP) of type `PMItemUPP` (page 136), using the function `NewPMItemUPP` (page 125) . You can do so with code like the following:

```
PMItemUPP MyPMItemCallbackUPP;
MyPMItemCallbackUPP = NewPMPItemUPP ( &MyPMItemCallback );
```

If you wish to call your event handling function directly, you can use the `InvokePMItemUPP` (page 125) function.

After you are finished with your event handling function, you can dispose of the UPP with the `DisposePMItemUPP` (page 126) function. However, if you will use the same event handling function for subsequent print jobs, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Special Considerations

Valid within the context of a printing session after creating a custom Page Setup or Print dialog box and calling `PMSetItemProc` (page 57) to register your dialog item event handling function.

Carbon Porting Notes

`PMItemProcPtr` is equivalent to the callback function address stored in the Classic Printing Manager's `TPrDlg.pItemProc` field.

Data Types

Printing Object Data Types

PMObject

```
typedef const void *                    PMObject;
```

Discussion
The base type for opaque Carbon Printing Manager objects.

PMDialog

Stores page setup and other printing settings.

```
typedef struct OpaquePMDialog*         PMDialog;
```

Discussion
Your application uses a reference to this opaque structure when working with custom Page Setup or Print dialog boxes. You create instances of this object using the `PMSessionPageSetupDialogInit` (page 52) or `PMSessionPrintDialogInit` (page 54) functions.

PMPrintSettings

Stores information such as the print quality and the range of pages to print for a particular printing session.

```
typedef struct OpaquePMPrintSettings*   PMPrintSettings;
```

Discussion
Your application uses this opaque object for storing print settings. Use the function `PMCreatePrintSettings` (page 40) to create an instance of this object for a particular printing session. When your application displays a Print dialog box by calling `PMSessionPrintDialog` (page 51) or `PMSessionPrintDialogMain` (page 55), the user can change these settings, which the Carbon Printing Manager stores in the `PMPrintSettings` object.

Although a `PMPrintSettings` object can be saved with a document, the object is intended to describe only one particular printing session.

Special Considerations
If you are using non-session printing functions, you should note that the `PMPrintSettings` object is valid only within the `PMBegin`/`PMEnd` block in which it is allocated, as the object is automatically disposed by the `PMEnd` function.

PMPageFormat

Stores information about how pages of a document should be printed.

```
typedef struct OpaquePMPageFormat*      PMPageFormat;
```

Discussion

The PMPageFormat object stores information such as page orientation, paper size, and printable area. Your application uses the function PMCreatePageFormat (page 38) to create an instance of this opaque object. When your application displays a Page Setup dialog box by calling PMSessionPageSetupDialog (page 50) or PMSessionPageSetupDialogMain (page 53), the user can change these settings, which the Carbon Printing Manager stores in the PMPageFormat object.

Although a PMPageFormat object can be saved with a document, the object is intended to describe only one particular printing session.

Special Considerations

If you are using non-session printing functions, you should note that the PMPageFormat object is valid only within the PMBegin/PMEnd block in which it is allocated, as the object is automatically disposed by the PMEnd function.

PMPrintContext

Describes the graphics environment for printing a document.

```
typedef struct OpaquePMPrintContext*    PMPrintContext;
```

Discussion

Your application uses this opaque object for imaging a print job. Use the function PMSessionBeginDocument (page 35) to create an instance, called a printing port, of this object. Your application then prints text and graphics by drawing into this printing port.

PMPrintSession

An opaque object representing a single printing session.

```
typedef struct OpaquePMPrintSession*    PMPrintSession;
```

Discussion

Your application uses a reference to this opaque structure to indentify a particular printing session. You create instances of this object using the `PMCreateSession` (page 34) function.

PMPrinter

An opaque object representing a printer.

```
typedef struct OpaquePMPrinter*        PMPrinter;
```

Discussion

Your application uses a reference to this opaque structure to indentify a particular printer. You create instances of this object using the `PMSessionGetCurrentPrinter` (page 62) function.

Printing Information Data Types

PMLanguageInfo

Contains level, version, and release information for the imaging language used by the printer driver.

```
struct PMLanguageInfo {
       Str32                    level;
       Str32                    version;
       Str32                    release;
   };
   typedef struct PMLanguageInfo PMLanguageInfo;
```

Field descriptions

`level`
: The level of the imaging language used by the printer driver.

`version`
: The version of the imaging language.

`release`
: The release of the imaging language.

Discussion

On return, the `PMPrinterGetLanguageInfo` (page 62) function supplies your application with level, version, and release information in the fields of this structure.

PMRect

Defines a rectangle using four double-point coordinates.

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

Field descriptions

`top`
: The vertical coordinate for the upper-left point of the rectangle.

`left`
: The horizontal coordinate for the upper-left point of the rectangle.

`bottom`
: The horizontal coordinate for the lower-right point of the rectangle.

`right`
: The vertical coordinate for the lower-right point of the rectangle.

PMResolution

Contains printing resolution information.

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

Field descriptions

`hRes`
: The horizontal resolution in dots per inch (dpi).

`vRes`
: The vertical resolution in dots per inch (dpi).

Callback Universal Procedure Pointer Data Types

PMIdleUPP

Defines a data type for the PMIdleProcPtr callback function pointer.

```
typedef PMIdleProcPtr PMIdleUPP;
```

Discussion
For a description of an idle callback function, see PMIdleProcPtr (page 126).

PMPageSetupDialogInitUPP

Defines a data type for the PMPageSetupDialogInitProcPtr callback function pointer.

```
typedef PMPageSetupDialogInitProcPtr PMPageSetupDialogInitUPP;
```

Discussion
For a description of a Page Setup dialog initialization callback function, see PMPageSetupDialogInitProcPtr (page 127).

PMPrintDialogInitUPP

Defines a data type for the PMPrintDialogInitProcPtr callback function pointer.

```
typedef PMPrintDialogInitProcPtr PMPrintDialogInitUPP;
```

Discussion
For a description of a Print dialog initialization callback function, see PMPrintDialogInitProcPtr (page 129).

PMItemUPP

Defines a data type for the PMItemProcPtr callback function pointer.

```
typedef PMItemProcPtr PMItemUPP;
```

Discussion
For a description of a dialog item event handling callback function, see PMItemProcPtr (page 130).

Constants

Color Mode Constants

Specify the color output mode for a print job.

```
typedef UInt16 PMColorMode;
   enum {
       kPMBlackAndWhite            = 1,
       kPMGray                     = 2,
       kPMColor                    = 3
   };
```

Constant descriptions

`kPMBlackAndWhite`
: Specifies black and white mode.

`kPMGray`
: Specifies grayscale mode.

`kPMColor`
: Specifies color mode.

Discussion

You use these constants with the `PMGetColorMode` (page 81) and `PMSetColorMode` (page 81) functions.

DestinationType Constants

Specify the output destination for a print job.

```
typedef UInt16 PMDestinationType;
   enum {
       kPMDestinationPrinter       = 1,
       kPMDestinationFile          = 2,
       kPMDestinationFax           = 3
   };
```

Constant descriptions

`kPMDestinationPrinter`
: Specifies output to a printer.

`kPMDestinationFile`
: Specifies output to a file.

`kPMDestinationFax`
: Specifies output to a fax.

Discussion

These constants are returned by the function `PMGetDestination` (page 83).

Document Format Strings

Specify the document format for a print job.

```
#define kPMDocumentFormatDefault
CFSTR("com.apple.documentformat.default")
#define kPMDocumentFormatPDF            CFSTR("application/pdf")
#define kPMDocumentFormatPICT           CFSTR("application/
vnd.apple.printing-pict")
#define kPMDocumentFormatPICTPS         CFSTR("application/
vnd.apple.printing-pict-ps")
#define kPMDocumentFormatPostScript     CFSTR("application/postscript")
```

Constant descriptions

`kPMDocumentFormatDefault`
: Specifies the default format for the current printer driver.

`kPMDocumentFormatPDF`
: Specifies PDF format.

`kPMDocumentFormatPICT`
: Specifies PICT format.

`kPMDocumentFormatPICTPS`
: Specifies PICT format with embedded PostScript.

`kPMDocumentFormatPostScript`
: Specifies PostScript format.

Discussion

You use these constants with the functions `PMSessionSetDocumentFormatGeneration` (page 84) and `PMSessionGetDocumentFormatSupported` (page 85).

Graphic Context Strings

Specify the graphics context for a print job.

```
   #define kPMGraphicsContextDefault
   CFSTR("com.apple.graphicscontext.default")
   #define kPMGraphicsContextQuickdraw
   CFSTR("com.apple.graphicscontext.quickdraw")
   #define kPMGraphicsContextCoreGraphics
   CFSTR("com.apple.graphicscontext.coregraphics")
```

Constant descriptions

`kPMGraphicsContextDefault`
: Specifies the default graphics context for the application's runtime environment. Currently this is QuickDraw for all Carbon applications.

`kPMGraphicsContextQuickdraw`
: Specifies a QuickDraw graphics context.

`kPMGraphicsContextCoreGraphics`
: Specifies a Core Graphics context. Not currently supported for Carbon applications.

Discussion

You use the constants with the `PMSessionSetDocumentFormatGeneration` (page 84) and `PMSessionGetGraphicsContext` (page 87) functions.

Page Orientation Constants

Specify page orientation.

```
typedef UInt16 PMOrientation;
   enum {
       kPMPortrait          = 1,
       kPMLandscape         = 2,
       kPMReversePortrait   = 3,
       PMReverseLandscape   = 4
   };
```

Constant descriptions

`kPMPortrait`
: Specifies portrait (vertical) page orientation. Portrait orientation performs no alteration of the logical page.

`kPMLandscape`

Specifies landscape (horizontal) page orientation. Landscape orientation performs a 90° clockwise rotation on the logical page rectangle and sets the origin to the lower left corner of the unrotated logical page. This has the effect of rotating the logical page image 90° counterclockwise.

`kPMReversePortrait`

Specifies reverse portrait page orientation for drivers that support this option. Drivers that do not support reverse portrait orientation will print in portrait mode. Reverse portrait orientation performs a 180° rotation on the logical page rectangle and sets the origin to the lower right corner of the unrotated logical page. This has the effect of rotating the logical page image 180°.

Reverse portrait orientation is seldom used and therefore not supported in the Page Setup dialog box. Although you may specify reverse portrait orientation for a document, you should be aware that the orientation will be changed to portrait (by default) or the user's selected orientation if your application displays the Page Setup dialog box and the user accepts the settings.

`kPMReverseLandscape`

Specifies reverse landscape page orientation for drivers that support this option. Drivers that do not support reverse landscape orientation will print in landscape mode. Reverse landscape orientation performs a 90° counterclockwise rotation on the logical page image and sets the origin to the upper right corner of the unrotated logical page. This has the effect of rotating the logical page image 90° clockwise.

Discussion

You use these constants with the `PMGetOrientation` (page 73) and `PMSetOrientation` (page 73) functions.

Tag Constants

Values that your application can pass to or obtain from Carbon Printing Manager functions that accept a tag paramter.

```
typedef UInt32 PMTag;
   enum {
       /* common tags */
```

```
        kPMCurrentValue           = FOUR_CHAR_CODE('curr'),
        kPMDefaultValue           = FOUR_CHAR_CODE('dflt'),
        kPMMinimumValue           = FOUR_CHAR_CODE('minv'),
        kPMMaximumValue           = FOUR_CHAR_CODE('maxv'),
        /* profile tags */
        kPMSourceProfile          = FOUR_CHAR_CODE('srcp'),
        /* resolution tags */
        kPMMinRange               = FOUR_CHAR_CODE('mnrg'),
        kPMMaxRange               = FOUR_CHAR_CODE('mxrg'),
        kPMMinSquareResolution    = FOUR_CHAR_CODE('mins'),
        kPMMaxSquareResolution    = FOUR_CHAR_CODE('maxs'),
        kPMDefaultResolution      = FOUR_CHAR_CODE('dftr')
    };
```

Constant descriptions

`kPMCurrentValue`
: Specifies the current setting or value.

`kPMDefaultValue`
: Specifies the default setting or value.

`kPMMinimumValue`
: Specifies the minimum setting or value.

`kPMMaximumValue`
: Specifies the maximum setting or value.

`kPMSourceProfile`
: Specifies the ColorSync source profile.

`kPMMinRange`
: Specifies the minimum resolution supported by the printer.

`kPMMaxRange`
: Specifies the maximum resolution supported by the printer.

`kPMMinSquareResolution`
: Specifies the minimum resolution setting for which the horizontal and vertical resolutions are equal.

`kPMMaxSquareResolution`
: Specifies the maximum resolution setting for which the horizontal and vertical resolutions are equal.

`kPMDefaultResolution`
: Specifies the default resolution setting for the printer (typically 72 dpi).

Discussion
You use these constants with the `PMPrinterGetPrinterResolution` (page 65), `PMSetProfile` (page 91), and `PMGetPrinterResolution` (page 112) functions.

kPMCancel

Indicates that the user canceled a printing operation.

```
#define kPMCancel                (0x0080)
```

Discussion
This constant is provided for compatibility with Classic applications and printer drivers that expect the `iPrAbort` error code to be returned when the user cancels a printing operation.

The Carbon Printing Manager's default idle function checks for Command-period keyboard events during printing, and sets the error condition equal to `kPMCancel` if one occurs. Your application can check for this condition using the `PMSessionError` function, and should cancel the print job if `kPMCancel` is returned.

If you supply your own idle function, it must check for Command-period keyboard events, and set the error condition using `PMSessionSetError( kPMCancel )`.

kPMPrintAllPages

Specifies that all pages of a document should be printed.

```
enum {
      kPMPrintAllPages            = -1
   };
```

Discussion
You can use this constant with the `PMSetLastPage` (page 78) and `PMSetPageRange` (page 80) functions.

kSizeOfTPrint

Specifies the size of the Classic Printing Manager `TPrint` structure.

```
enum {
       kSizeOfTPrint = 120
   };
```

Discussion

Your application can use this constant when allocating memory for a Classic Printing Manager print record.

Unwanted Data Constants

Values your application can use to indicate it does not need certain types of data returned by the Carbon Printing Manager.

```
   #define kPMNoData            NULL
   #define kPMDontWantSize      NULL
   #define kPMDontWantData      NULL
   #define kPMDontWantBoolean   NULL
   #define kPMNoPrintSettings   NULL
   #define kPMNoPageFormat      NULL
   #define kPMNoReference       NULL
```

Constant descriptions

`kPMNoData`

Specifies that your application does not need data returned for a particular parameter. For future compatibility, you are encouraged to use one of the following constants in cases where a specific type of data is not required.

`kPMDontWantSize`

Specifies that your application does not need the size information returned by the Carbon Printing Manager.

`kPMDontWantData`

Specifies that your application does not need the data returned by the Carbon Printing Manager.

`kPMDontWantBoolean`

Specifies that your application does not need a Boolean value returned by the Carbon Printing Manager.

`kPMNoPrintSettings`
: Specifies that your application does not need a `PMPrintSettings` object returned by the Carbon Printing Manager.

`kPMNoPageFormat`
: Specifies that your application does not need a `PMPageFormat` object returned by the Carbon Printing Manager.

`kPMNoReference`
: Specifies that your application does not need an address pointer returned by the Carbon Printing Manager.

Result Codes

The most common result codes returned by the Carbon Printing Manager are listed below.

Result Code	Value	Description
`kPMNoError`	0	No error.
`kPMInvalidParameter`	-50	A required parameter is missing or invalid.
`kPMGeneralError`	-30870	An unspecific error occurred.
`kPMOutOfScope`	-30871	Your application called this function out of sequence with other Carbon Printing Manager functions.
`kPMNoDefaultPrinter`	-30872	The user has not specified a default printer.
`kPMNotImplemented`	-30873	This function is not implemented.
`kPMNoSuchEntry`	-30874	There is no entry to match your application request.
`kPMInvalidPrintSettings`	-30875	Your application passed an invalid `PMPrintSettings` object.
`kPMInvalidPageFormat`	-30876	Your application passed an invalid `PMPageFormat` object.
`kPMValueOutOfRange`	-30877	Your application passed an out-of-range value.
`kPMLockIgnored`	-30878	The lock value was ignored.

Result Code	Value	Description
`kPMInvalidPrintSession`	-30879	Your application passed an invalid `PMPrintSession` object.
`kPMInvalidPrinter`	-30880	Your application passed an invalid `PMPrinter` object.
`kPMObjectInUse`	-30881	The specified object is in use.

A P P E N D I X   A

Sample Code

Using Session Functions

```
/*---------------------------------------------------------------------------------

    This sample code is the Carbon equivalent of the Classic print loop
    documented in Tech Note 1092 “A Print Loop That Cares ...”.  This code
    illustrates the use of the Carbon Printing Manager’s session APIs.

    You may incorporate this sample code into your applications without
    restriction, though the sample code has been provided “AS IS” and the
    responsibility for its operation is 100% yours.  However, what you are
    not permitted to do is to redistribute the source as “Apple Sample Code”
    after having made changes. If you’re going to re-distribute the source,
    we require that you make it clear in the source that the code was
    descended from Apple Sample Code, but that you’ve made changes.

    Copyright © 1998-1999 Apple Computer, Inc., All Rights Reserved

    Change History:
        17 Oct 99    pd  Updated for Carbon 1.1 SDK to use session APIs
        13 Sep 99    pd  Updated for Carbon 1.0 SDK
        11 Apr 99    dk  Original version for Developer Preview 1

---------------------------------------------------------------------------------*/

#define TARGET_API_MAC_CARBON 1
```

```
/*------------------------------------------------------------------------------
    Includes
------------------------------------------------------------------------------*/
#include <Quickdraw.h>
#include <Fonts.h>
#include <Resources.h>
#include <PMApplication.h>
#include <NumberFormatting.h>    // for NumToString


/*------------------------------------------------------------------------------
    Globals
------------------------------------------------------------------------------*/
short   gAppResFile = 0;         // application’s resource file
Handle  gflatPageFormat = nil;   // used in FlattenAndSavePageFormat


/*------------------------------------------------------------------------------
    Prototypes
------------------------------------------------------------------------------*/
int         main(void);
OSStatus    DoPageSetupDialog(PMPrintSession printSession, PMPageFormat* pageFormat);
OSStatus    DoPrintDialog(PMPrintSession printSession, PMPageFormat pageFormat,
                PMPrintSettings* printSettings);
void        DoPrintLoop(PMPrintSession printSession, PMPageFormat pageFormat,
                PMPrintSettings printSettings);
OSStatus    FlattenAndSavePageFormat(PMPageFormat pageFormat);
OSStatus    LoadAndUnflattenPageFormat(PMPageFormat* pageFormat);
UInt32      DetermineNumberOfPagesInDoc(PMRect rect);
void        DrawPage(PMRect pageRect, UInt32 pageNumber);
pascal void pIdleProc(void);
void        PostPrintingErrors(OSStatus status);


/*------------------------------------------------------------------------------
    Function: main

    Parameters:
        <none>
    Description:
        Uses PMCreateSession/PMRelease instead of PMBegin/PMEnd.  Note that the two
        printing objects, PMPageSetup and PMPrintSettings are valid outside the
        printing session.  This is not the case with PMBegin/PMEnd.  Note also
        that no nesting of printing sessions is allowed for Carbon applications
        running under MacOS 8 or 9.
        In this sample code, we show where an old print record could be read in and
        converted.  This is followed by displaying the Page Setup dialog, then the
        Print dialog, and finally we print the page(s) of the document.

------------------------------------------------------------------------------*/
int main(void)
{
    OSStatus         status = noErr;
    GrafPtr          oldPort;
    PMPageFormat     pageFormat = kPMNoPageFormat;
    PMPrintSettings  printSettings = kPMNoPrintSettings;
    PMPrintSession   printSession;
    Handle           gOldPrintRecord = nil;   // placeholder for old print record

    //  Standard Toolbox initialization.
    InitCursor();

    //  Save the old QD grafport.
    GetPort(&oldPort);

    //  Save the current resource file so that it can be used in pIdleProc.
    gAppResFile = CurResFile();

    //  Initialize the printing manager and create a printing session.
    status = PMCreateSession(&printSession);
    if (status != noErr) return 0;   // pointless to continue if PMCreateSession fails

    //  If your application has an old print record, it can be converted into new
    //  PMPageFormat and PMPrintSettings objects.  In this sample code, we skip this
    //  step.
    if (gOldPrintRecord != nil)
        status = PMSessionConvertOldPrintRecord(printSession, gOldPrintRecord,
                    &printSettings, &pageFormat);
    //  Display the Page Setup dialog.
    if (status == noErr)
        status = DoPageSetupDialog(printSession, &pageFormat);

    //  Display the Print dialog.
    if (status == noErr)
        status = DoPrintDialog(printSession, pageFormat, &printSettings);

    //  Execute the print loop.
    if (status == noErr)
        DoPrintLoop(printSession, pageFormat, printSettings);

    //  Release the PageFormat and PrintSettings objects.  PMRelease decrements the
    //  ref count of the allocated objects.  We let the Printing Manager decide when
    //  to release the allocated memory.
    if (pageFormat != kPMNoPageFormat)
        (void)PMRelease(&pageFormat);
    if (printSettings != kPMNoPrintSettings)
        (void)PMRelease(&printSettings);

    //  Terminate the current printing session.
    (void)PMRelease(&printSession);

    //  Restore the original QD grafport.
    SetPort(oldPort);

    return 0;

}   // main


/*------------------------------------------------------------------------------
    Function: DoPageSetupDialog

    Parameters:
        printSession     -   current printing session
        pageFormat       -   a PageFormat object addr

    Description:
        If the caller passes in an empty PageFormat object, create a new one,
        otherwise validate the one provided by the caller.
        Invokes the Page Setup dialog and checks for Cancel.
        Flattens the PageFormat object so it can be saved with the document.
        Note that the PageFormat object is modified by this function.

-----------------------------------------------------------------------------*/
OSStatus DoPageSetupDialog(PMPrintSession printSession, PMPageFormat* pageFormat)
{
    OSStatus    status;
    Boolean     accepted;

    //  Set up a valid PageFormat object.
    if (*pageFormat == kPMNoPageFormat)
    {
        status = PMCreatePageFormat(pageFormat);

        //  Note that PMPageFormat is not session-specific, but calling
        //  PMSessionDefaultPageFormat assigns values specific to the printer
        //  associated with the current printing session.
        if ((status == noErr) && (*pageFormat != kPMNoPageFormat))
            status = PMSessionDefaultPageFormat(printSession, *pageFormat);
    }
    else
        status = PMSessionValidatePageFormat(printSession, *pageFormat,
                    kPMDontWantBoolean);

    //  Display the Page Setup dialog.
    if (status == noErr)
    {
        status = PMSessionPageSetupDialog(printSession, *pageFormat, &accepted);
        if (!accepted)
            status = kPMCancel;  // user clicked Cancel button
    }

    //  If the user did not cancel, flatten and save the PageFormat object
    //  with our document.
    if (status == noErr)
        status = FlattenAndSavePageFormat(*pageFormat);

    return status;

}   //  DoPageSetupDialog
```

```
/*------------------------------------------------------------------------------
    Function: DoPrintDialog

    Parameters:
        printSession     -   current printing session
        pageFormat       -   a PageFormat object addr
        printSettings    -   a PrintSettings object addr

    Description:
        If the caller passes an empty PrintSettings object, create a new one,
        otherwise validate the one provided by the caller.
        Invokes the Print dialog and checks for Cancel.
        Note that the PrintSettings object is modified by this function.

------------------------------------------------------------------------------*/
OSStatus DoPrintDialog(PMPrintSession printSession, PMPageFormat pageFormat,
                PMPrintSettings* printSettings)
{
    OSStatus    status;
    Boolean     accepted;
    UInt32      minPage = 1,
                maxPage = 9999;

    //  In this sample code the caller provides a valid PageFormat reference but in
    //  your application you may want to load and unflatten the PageFormat object
    //  that was saved at PageSetup time.  See LoadAndUnflattenPageFormat below.

    //  Set up a valid PrintSettings object.
    if (*printSettings == kPMNoPrintSettings)
    {
        status = PMCreatePrintSettings(printSettings);

        //  Note that PMPrintSettings is not session-specific, but calling
        //  PMSessionDefaultPrintSettings assigns values specific to the printer
        //  associated with the current printing session.
        if ((status == noErr) && (*printSettings != kPMNoPrintSettings))
            status = PMSessionDefaultPrintSettings(printSession, *printSettings);
    }
    else
        status = PMSessionValidatePrintSettings(printSession, *printSettings,
                    kPMDontWantBoolean);
    //  Set a valid page range before displaying the Print dialog
    if (status == noErr)
        status = PMSetPageRange(*printSettings, minPage, maxPage);

    //  Display the Print dialog.
    if (status == noErr)
    {
        status = PMSessionPrintDialog(printSession, *printSettings, pageFormat,
                    &accepted);
        if (!accepted)
            status = kPMCancel;  // user clicked Cancel button
    }

    return status;

}   //  DoPrintDialog

/*------------------------------------------------------------------------------
    Function: DoPrintLoop

    Parameters:
        printSession    -   current printing session
        pageFormat      -   a PageFormat object addr
        printSettings   -   a PrintSettings object addr

    Description:
        Assumes the caller provides validated PageFormat and PrintSettings
        objects.
        Calculates a valid page range and prints each page by calling DrawPage.

------------------------------------------------------------------------------*/
void    DoPrintLoop(PMPrintSession printSession, PMPageFormat pageFormat,
            PMPrintSettings printSettings)
{
    OSStatus    status,
                printError;
    PMRect      pageRect;
    UInt32      realNumberOfPagesinDoc,
                pageNumber,
                firstPage,
                lastPage;
```

```
//  PMGetAdjustedPaperRect returns the paper size taking into account rotation,
//  resolution, and scaling settings.  Note this is the paper size selected
//  the Page Setup dialog.  It is not guaranteed to be the same as the paper
//  size selected in the Print dialog on Mac OS X.
status = PMGetAdjustedPaperRect(pageFormat, &pageRect);

//  PMGetAdjustedPageRect returns the page size taking into account rotation,
//  resolution, and scaling settings.  Note this is the imageable area of the
//  paper selected in the Page Setup dialog.
//  DetermineNumberOfPagesInDoc returns the number of pages required to print
//  the document.
if (status == noErr)
{
    status = PMGetAdjustedPageRect(pageFormat, &pageRect);
    if (status == noErr)
        realNumberOfPagesinDoc = DetermineNumberOfPagesInDoc(pageRect);
}

//  Get the user's selection for first and last pages
if (status == noErr)
{
    status = PMGetFirstPage(printSettings, &firstPage);
    if (status == noErr)
        status = PMGetLastPage(printSettings, &lastPage);
}

//  Check that the selected page range does not go beyond the actual
//  number of pages in the document.
if (status == noErr)
{
    if (realNumberOfPagesinDoc < lastPage)
        lastPage = realNumberOfPagesinDoc;
}

//  Tell the Printing Manager to call our idle proc.
if (status == noErr)
    status = PMSessionSetIdleProc(printSession, NewPMIdleProc(pIdleProc));

//  NOTE:  We don't have to worry about the number of copies.  The Printing
//  Manager handles this.  So we just iterate through the document from the
//  first page to be printed, to the last.
```

```
if (status == noErr)
{
    //  Establish a graphics context for drawing the document's pages.
    //  Although it's not used in this sample code, PMGetGrafPtr can be called
    //  get the QD grafport.
    status = PMSessionBeginDocument(printSession, printSettings, pageFormat);
    if (status == noErr)
    {
        //  Print all the pages in the document.  Note that we spool all pages
        //  and rely upon the Printing Manager to print the correct page range.
        //  In this sample code we assume the total number of pages in the
        //  document is equal to "lastPage".
        pageNumber = 1;
        while ((pageNumber <= lastPage) && (PMSessionError() == noErr))
        {
            //  NOTE:  We don't have to deal with the old Printing Manager's
            //  128-page boundary limit anymore.

            //  Set up a page for printing.
            status = PMSessionBeginPage(printSession, pageFormat, &pageRect);
            if (status != noErr)
                break;

            //  Draw the page.
            DrawPage(pageRect, pageNumber);

            //  Close the page.
            status = PMSessionEndPage(printSession);
            if (status != noErr)
                break;

            //  And loop.
            pageNumber++;
        }

        // Close the printing port
        (void)PMSessionEndDocument(printSession);
    }
}
```

```
    //  Only report a printing error once we have completed the print loop. This
    //  ensures that every PMSessionBegin... call is followed by a matching
    //  PMSessionEnd... call, so the Printing Manager can release all temporary
    //  memory and close properly.
    printError = PMSessionError();
    if (printError != noErr)
        PostPrintingErrors(printError);

}   //  DoPrintLoop

/*------------------------------------------------------------------------------
    Function: FlattenAndSavePageFormat

    Parameters:
        pageFormat   -   a PageFormat object

    Description:
        Flattens a PageFormat object so it can be saved with the document.
        Assumes caller passes a validated PageFormat object.

------------------------------------------------------------------------------*/
OSStatus FlattenAndSavePageFormat(PMPageFormat pageFormat)
{
    OSStatusstatus;
    Handle   flatFormatHandle = nil;

    //  Flatten the PageFormat object to memory.
    status = PMFlattenPageFormat(pageFormat, &flatFormatHandle);

    //  Write the PageFormat data to file.
    //  In this sample code we simply copy it to a global.
    gflatPageFormat = flatFormatHandle;

    return status;
}   //  FlattenAndSavePageFormat

/*------------------------------------------------------------------------------
    Function: LoadAndUnflattenPageFormat

    Parameters:
        pageFormat   -   a PageFormat object read from a document file
    Description:
        Gets flattened PageFormat data from the document and returns a PageFormat
        object. The function is not called in this sample code but your application
        will need to retrieve PageFormat data saved with documents.

-------------------------------------------------------------------------------*/
OSStatus LoadAndUnflattenPageFormat(PMPageFormat* pageFormat)
{
    OSStatus    status;
    Handle      flatFormatHandle = nil;

    //  Read the PageFormat flattened data from file.
    //  In this sample code we simply copy it from a global.
    flatFormatHandle = gflatPageFormat;

    //  Convert the PageFormat flattened data into a PageFormat object.
    status = PMUnflattenPageFormat(flatFormatHandle, pageFormat);

    return status;
}   //  LoadAndUnflattenPageFormat


/*-------------------------------------------------------------------------------
    Function: DetermineNumberOfPagesInDoc

    Parameters:
        pageRect    -   size of the document, from PMGetAdjustedPageRect

    Description:
        Compares the size of the document with the number of pages in the
        PageFormat object to calculate the number of pages required to print
        the document.

-------------------------------------------------------------------------------*/
UInt32 DetermineNumberOfPagesInDoc(PMRect pageRect)
{
#pragma unused (pageRect)
    //  In this sample code we simply return a hard coded number.
    return 1;
}   //  DetermineNumberOfPagesinDoc
```

```
/*------------------------------------------------------------------------------
    Function: DrawPage

    Parameters:
        pageRect     -   size of the page we're drawing
        pageNumber   -   the logical page number in the document

    Description:
        Draws the contents of a single page.

------------------------------------------------------------------------------*/
void DrawPage(PMRect pageRect, UInt32 pageNumber)
{
    Str255      pageString;

    //  In this sample code we simply write the word "Page" followed by the
    //  page number onto each page.

    MoveTo((short)(pageRect.left + (pageRect.right - pageRect.left) / 2),
        (short)(pageRect.bottom - (pageRect.bottom - pageRect.top) / 2));
    DrawString("\pPage ");
    NumToString((long)pageNumber, pageString);
    DrawString(pageString);
}   //  DrawPage


/*------------------------------------------------------------------------------
    Function: pIdleProc

    Parameters:
        <none>

    Description:
        This routine is called by the Printing Manager while the print job is
        being spooled (background printing) or printed (foreground printing).
        Since most print drivers display their own progress dialog boxes, there is
        little need for applications to use the idle proc for progress reporting.

------------------------------------------------------------------------------*/
pascal void pIdleProc(void)
{
    //  Save the current resource file.
    short   curRes = CurResFile();

    //  Set the application's resource file.
    UseResFile(gAppResFile);

    //  Do something useful.

    //  Restore the resource file.
    UseResFile(curRes);

}   //  pIdleProc


/*------------------------------------------------------------------------------
    Function: PostPrintingErrors

    Parameters:
        status  -   error code

    Description:
        This is where we could post an alert to report any problem encountered
        by the Printing Manager.

------------------------------------------------------------------------------*/
void PostPrintingErrors(OSStatus status)
{
#pragma unused (status)
}   //  PostPrintingErrors
```

Using Non-Session Functions

```
/*------------------------------------------------------------------------------

    This sample code is the Carbon equivalent of the Classic print loop
    documented in Tech Note 1092 "A Print Loop That Cares ...".  This code
    illustrates the use of the Carbon Printing Manager's non-session APIs.
    You may incorporate this sample code into your applications without
    restriction, though the sample code has been provided “AS IS” and the
    responsibility for its operation is 100% yours.  However, what you are
    not permitted to do is to redistribute the source as “Apple Sample Code”
    after having made changes. If you’re going to re-distribute the source,
    we require that you make it clear in the source that the code was
    descended from Apple Sample Code, but that you’ve made changes.

    Copyright © 1998-1999 Apple Computer, Inc., All Rights Reserved

    Change History:
        13 Sep 99    pd  Updated for Carbon 1.0 SDK
        11 Apr 99    dk  Original version for Developer Preview 1

------------------------------------------------------------------------------*/

#defineTARGET_API_MAC_CARBON 1

/*------------------------------------------------------------------------------
    Includes
------------------------------------------------------------------------------*/
#include <Quickdraw.h>
#include <Fonts.h>
#include <Resources.h>
#include <PMApplication.h>
#include <NumberFormatting.h>    // for NumToString

/*------------------------------------------------------------------------------
    Globals
------------------------------------------------------------------------------*/
short   gAppResFile = 0;         // application’s resource file
Handle  gflatPageFormat = nil;   // used in FlattenAndSavePageFormat

/*------------------------------------------------------------------------------
    Prototypes
------------------------------------------------------------------------------*/
int         main(void);
OSStatus    DoPageSetupDialog(PMPageFormat* pageFormat);
OSStatus    DoPrintDialog(PMPageFormat* pageFormat, PMPrintSettings* printSettings);
void        DoPrintLoop(PMPageFormat* pageFormat, PMPrintSettings* printSettings);
OSStatus    FlattenAndSavePageFormat(PMPageFormat pageFormat);
OSStatus     LoadAndUnflattenPageFormat(PMPageFormat* pageFormat);
UInt32       DetermineNumberOfPagesInDoc(PMRect rect);
void         DrawPage(PMRect pageRect, PMPrintContext printingPort, UInt32 pageNumber);
pascal void  pIdleProc(void);
void         PostPrintingErrors(OSStatus status);

/*------------------------------------------------------------------------------
    Function: main

    Parameters:
        <none>

    Description:
        Contains the PMBegin/PMEnd block, within which all other Printing Manager
        calls are made, and within which the PMPageSetup and PMPrintSettings objects
        are valid.
        Displays a Page Setup dialog, then the Print dialog, then executes a simple
        print loop.

------------------------------------------------------------------------------*/
int main(void)
{
    OSStatus          status = noErr;
    GrafPtr           oldPort;
    PMPageFormat      pageFormat = kPMNoPageFormat;
    PMPrintSettings   printSettings = kPMNoPrintSettings;
    Handle            gOldPrintRecord = nil;   // placeholder for old print record

    //  Standard Toolbox initialization.
    InitCursor();

    //  Save the old QD grafport.
    GetPort(&oldPort);

    //  Save the current resource file so that it can be used in pIdleProc.
    gAppResFile = CurResFile();

    //  Initialize the printing manager. Now a required call.
    status = PMBegin();
    if (status != noErr) return 0;   // pointless to continue if PMBegin fails
    //  If your application has an old print record, it can be converted into new
    //  PMPageFormat and PMPrintSettings objects.  In this sample code, we skip this
    //  step.
    if (gOldPrintRecord != nil)
        status = PMConvertOldPrintRecord(gOldPrintRecord, &printSettings, &pageFormat);

    //  Display the Page Setup dialog.
    if (status == noErr)
        status = DoPageSetupDialog(&pageFormat);

    //  Display the Print dialog.
    if (status == noErr)
        status = DoPrintDialog(&pageFormat, &printSettings);

    //  Execute the print loop.
    if (status == noErr)
        DoPrintLoop(&pageFormat, &printSettings);

    //  Deallocate the PageFormat and PrintSettings objects.
    if (pageFormat != kPMNoPageFormat)
        {
        (void)PMDisposePageFormat(pageFormat);
        pageFormat = kPMNoPageFormat;
        }

    if (printSettings != kPMNoPrintSettings)
        {
        (void)PMDisposePrintSettings(printSettings);
        printSettings = kPMNoPrintSettings;
        }

    //  Close the printing manager. Now a required call.
    (void)PMEnd();

    //  Restore the original QD grafport.
    SetPort(oldPort);

    return 0;

}   // main
```

```
/*------------------------------------------------------------------------------
    Function: DoPageSetupDialog

    Parameters:
        pageFormat   -   a PageFormat object addr

    Description:
        If the caller passes in an empty PageFormat object, create a new one,
        otherwise validate the one provided by the caller.
        Invokes the Page Setup dialog and checks for Cancel.
        Flattens the PageFormat object so it can be saved with the document.
        Note that the PageFormat object is modified by this function.

------------------------------------------------------------------------------*/
OSStatus DoPageSetupDialog(PMPageFormat* pageFormat)
{
    OSStatus    status;
    Boolean     accepted;

    //  Set up a valid PageFormat object.
    if (*pageFormat == kPMNoPageFormat)
    {
        status = PMNewPageFormat(pageFormat);

        if ((status == noErr) && (*pageFormat != kPMNoPageFormat))
            status = PMDefaultPageFormat(*pageFormat);
    }
    else
        status = PMValidatePageFormat(*pageFormat, kPMDontWantBoolean);

    //  Display the Page Setup dialog.
    if (status == noErr)
    {
        status = PMPageSetupDialog(*pageFormat, &accepted);
        if (!accepted)
            status = kPMCancel;  // user clicked Cancel button
    }

    //  If the user did not cancel, flatten and save the PageFormat object
    //  with our document.
    if (status == noErr)
```

```
        status = FlattenAndSavePageFormat(*pageFormat);

    return status;

}   //  DoPageSetupDialog


/*------------------------------------------------------------------------------
    Function: DoPrintDialog

    Parameters:
        pageFormat       -   a PageFormat object addr
        printSettings    -   a PrintSettings object addr

    Description:
        If the caller passes an empty PrintSettings object, create a new one,
        otherwise validate the one provided by the caller.
        Invokes the Print dialog and checks for Cancel.
        Note that the PrintSettings object is modified by this function.

------------------------------------------------------------------------------*/
OSStatus DoPrintDialog(PMPageFormat* pageFormat, PMPrintSettings* printSettings)
{
    OSStatus    status;
    Boolean     accepted;

    //  In this sample code the caller provides a valid PageFormat reference but in
    //  your application you may want to load and unflatten the PageFormat object
    //  that was saved at PageSetup time.  See LoadAndUnflattenPageFormat below.

    //  Set up a valid PrintSettings object.
    if (*printSettings == kPMNoPrintSettings)
    {
        status = PMNewPrintSettings(printSettings);

        if ((status == noErr) && (*printSettings != kPMNoPrintSettings))
            status = PMDefaultPrintSettings(*printSettings);
    }
    else
        status = PMValidatePrintSettings(*printSettings, kPMDontWantBoolean);
    //  Display the Print dialog.
    if (status == noErr)
    {
        status = PMPrintDialog(*printSettings, *pageFormat, &accepted);
        if (!accepted)
            status = kPMCancel;  // user clicked Cancel button
    }

    return status;

}   //  DoPrintDialog


/*------------------------------------------------------------------------------
    Function: DoPrintLoop

    Parameters:
        pageFormat      -   a PageFormat object addr
        printSettings   -   a PrintSettings object addr

    Description:
        Depends on the caller providing validated PageFormat and PrintSettings
        objects. Calculates a valid page range and prints each page by calling DrawPage.

------------------------------------------------------------------------------*/
void DoPrintLoop(PMPageFormat* pageFormat, PMPrintSettings* printSettings)
{
    OSStatus        status,
                    printError;
    PMRect          pageRect;
    PMPrintContext  thePrintingPort = kPMNoReference;
    UInt32          realNumberOfPagesinDoc,
                    pageNumber,
                    firstPage,
                    lastPage;

    //  PMGetAdjustedPaperRect returns the paper size taking into account rotation,
    //  resolution and scaling settings.  This is one example of the many accessor
    //  functions provided by the Carbon Printing Manager.
    status = PMGetAdjustedPaperRect(*pageFormat, &pageRect);
```

```
//  PMGetAdjustedPageRect returns the page size taking into account rotation,
//  resolution and scaling settings.  DetermineNumberOfPagesInDoc returns the
//  number of pages required to print the document.
if (status == noErr)
{
    status = PMGetAdjustedPageRect(*pageFormat, &pageRect);
    if (status == noErr)
        realNumberOfPagesinDoc = DetermineNumberOfPagesInDoc(pageRect);
}

//  Get the user's selection for first and last pages
if (status == noErr)
{
    status = PMGetFirstPage(*printSettings, &firstPage);
    if (status == noErr)
        status = PMGetLastPage(*printSettings, &lastPage);
}

//  Check that the selected page range does not go beyond the actual
//  number of pages in the document.
if (status == noErr)
{
    if (realNumberOfPagesinDoc < lastPage)
        lastPage = realNumberOfPagesinDoc;
}

//  Tell the Printing Manager to call our idle proc.
if (status == noErr)
    status = PMSetIdleProc(NewPMIdleProc(pIdleProc));

//  NOTE:  We don't have to worry about the number of copies.  The Printing
//  Manager handles this.  So we just iterate through the document from the
//  first page to be printed, to the last.
if (status == noErr)
{
    //  Establish a printing port for drawing.
    status = PMBeginDocument(*printSettings, *pageFormat, &thePrintingPort);
    if ((status == noErr) && (thePrintingPort != kPMNoReference))
    {
        //  Print the selected range of pages in the document.
        pageNumber = firstPage;
```

```
            while ((pageNumber <= lastPage) && (PMError() == noErr))
            {
                //  NOTE:  We don’t have to deal with the old Printing Manager’s
                //  128-page boundary limit anymore.

                //  Establish a printing page.
                status = PMBeginPage(thePrintingPort, nil);
                if (status != noErr)
                    break;

                //  Draw the page.
                DrawPage(pageRect, thePrintingPort, pageNumber);

                //  Close the page.
                status = PMEndPage(thePrintingPort);
                if (status != noErr)
                    break;

                //  And loop.
                pageNumber++;
            }

            // Close the printing port
            (void)PMEndDocument(thePrintingPort);
        }
    }

    //  Only report a printing error once we have completed the print loop. This
    //  ensures that every PMBegin... call is followed by a matching PMEnd...
    //  call, so the Printing Manager can release all temporary memory and close
    //  properly.
    printError = PMError();
    if (printError != noErr)
        PostPrintingErrors(printError);

}   //  DoPrintLoop
```

```
/*------------------------------------------------------------------------------
    Function: FlattenAndSavePageFormat

    Parameters:
        pageFormat   -   a PageFormat object

    Description:
        Flattens a PageFormat object so it can be saved with the document.
        Assumes caller passes a validated PageFormat object.

------------------------------------------------------------------------------*/
OSStatus FlattenAndSavePageFormat(PMPageFormat pageFormat)
{
    OSStatus    status;
    Handle      flatFormatHandle = nil;

    //  Flatten the PageFormat object to memory.
    status = PMFlattenPageFormat(pageFormat, &flatFormatHandle);

    //  Write the PageFormat data to file.
    //  In this sample code we simply copy it to a global.
    gflatPageFormat = flatFormatHandle;

    return status;
}   //  FlattenAndSavePageFormat



/*------------------------------------------------------------------------------
    Function: LoadAndUnflattenPageFormat

    Parameters:
        pageFormat   -   PageFormat object read from document file

    Description:
        Gets flattened PageFormat data from the document and returns a PageFormat
        object. The function is not called in this sample code but your application
        will need to retrieve PageFormat data saved with documents.

------------------------------------------------------------------------------*/
```

```
OSStatus LoadAndUnflattenPageFormat(PMPageFormat* pageFormat)
{
    OSStatus    status;
    Handle      flatFormatHandle = nil;

    //  Read the PageFormat flattened data from file.
    //  In this sample code we simply copy it from a global.
    flatFormatHandle = gflatPageFormat;

    //  Convert the PageFormat flattened data into a PageFormat object.
    status = PMUnflattenPageFormat(flatFormatHandle, pageFormat);

    return status;
}   //  LoadAndUnflattenPageFormat


/*------------------------------------------------------------------------------
    Function: DetermineNumberOfPagesInDoc

    Parameters:
        pageRect    -   size of the document, from PMGetAdjustedPageRect

    Description:
        Compares the size of the document with the number of pages in the
        PageFormat object to calculate the number of pages required to print
        the document.

------------------------------------------------------------------------------*/
UInt32DetermineNumberOfPagesInDoc(PMRect pageRect)
{
#pragma unused (pageRect)
    //  In this sample code we simply return a hard coded number.
    return 1;
}   //  DetermineNumberOfPagesinDoc


/*------------------------------------------------------------------------------
    Function: DrawPage

    Parameters:
        pagerect    -   size of the page we're drawing
        printingPort -   the page’s grafport
        pageNumber   -   the logical page number in the document

    Description:
        Draws the contents of a single page.

--------------------------------------------------------------------------------*/
void DrawPage(PMRect pageRect, PMPrintContext printingPort, UInt32 pageNumber)
{
#pragma unused (printingPort)
    Str255      pageString;

    //  In this sample code we simply write the word “Page” followed by the
    //  page number onto each page.

    MoveTo((short)(pageRect.left + (pageRect.right - pageRect.left) / 2),
        (short)(pageRect.bottom - (pageRect.bottom - pageRect.top) / 2));
    DrawString(“\pPage “);
    NumToString((long)pageNumber, pageString);
    DrawString(pageString);

}   //  DrawPage


/*--------------------------------------------------------------------------------
    Function: pIdleProc

    Parameters:
        <none>

    Description:
        This routine is called by the Printing Manager while the print job is
        being spooled (background printing) or printed (foreground printing).
        Since most print drivers display their own progress dialog boxes, there is
        little need for applications to use the idle proc for progress reporting.

--------------------------------------------------------------------------------*/
pascal void pIdleProc(void)
{
    //  Save the current resource file.
    short   curRes = CurResFile();
    //  Set the application's resource file.
    UseResFile(gAppResFile);

    //  Do something useful.

    //  Restore the resource file.
    UseResFile(curRes);

}   //  pIdleProc



/*------------------------------------------------------------------------------
    Function: PostPrintingErrors

    Parameters:
        status   -   error code

    Description:
        This is where we could post an alert to report any problem encountereds
        by the Printing Manager.

------------------------------------------------------------------------------*/
void PostPrintingErrors(OSStatus status)
{
#pragma unused (status)
}   //  PostPrintingErrors
```