Extending Printing Dialogs in Mac OS X

Preliminary

5/23/00

© Apple Computer, Inc. 2000

Preliminary © Apple Computer, Inc. 5/23/00

Apple Computer, Inc.
© 1999 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except to make a backup copy of any documentation provided on CD-ROM.

The Apple logo is a trademark of Apple Computer, Inc.
Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Macintosh, and WebObjects are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Enterprise Objects is a trademark of Apple Computer, Inc.

NeXT, the NeXT logo, OPENSTEP, Enterprise Objects Framework, Objective–C, and WEBSCRIPT are trademarks of NeXT Software, Inc.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

ORACLE is a registered trademark of Oracle Corporation, Inc.

SYBASE is a registered trademark of Sybase, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Windows NT is a trademark of Microsoft Corporation.

All other trademarks mentioned belong to their respective owners.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Preliminary © Apple Computer, Inc. 5/23/00

Contents

Preliminary © Apple Computer, Inc. 5/23/00

Preliminary © Apple Computer, Inc. 5/23/00

Figures, Listings, and Tables

Preliminary © Apple Computer, Inc. 5/23/00

Preliminary © Apple Computer, Inc. 5/23/00

Extending Printing Dialogs in Mac OS X

Note: This is a preliminary document. While it has received some technical review, there are likely to be changes to some of the information provided here.

This document describes how to use the Mac OS X Printing Manager's printing dialog extension (PDE) mechanism to add application-specific information or controls to the Print and Page Setup dialogs. Although much of that information can also be applied to printer module PDEs, detailed coverage of printer modules is not provided here.

This document provides overview material, code samples, and API reference, in the following sections:

The sample code in this document is taken from a complete implementation of a PDE which will be distributed with the Mac OS X Printing Manager SDK. The sample code is for Carbon applications.

Preliminary © Apple Computer, Inc. 5/23/00

Extending Printing Dialogs in Mac OS X

This document assumes you are familiar with the Classic Printing Manager and printing concepts in general. You should also be familiar with the Control Manager and with Mac OS APIs that provide cut and paste, drag and drop, high-level events, and Apple events.

The following documents contain information that may be useful in working with PDEs. All are available at the Apple website at:

<http://developer.apple.com/techpubs/carbon>

- Inside Mac OS X: System Overview (the "System Architecture" chapter provides an overview of the Mac OS X printing system)
- Adopting the Carbon Printing Manager
- Carbon Printing Manager Reference
- Adopting the Aqua Interface
- Introduction to the Carbon Event Manager
- Core Foundation Plug-In Services
- Core Foundation Bundle Services

Note: PDEs are not supported in the DP4 release of Mac OSX, though they are scheduled for inclusion in the 1.0 release. Also scheduled for the 1.0 release are:

1. Printing help available through a help button on printing dialogs.

2. Saved settings (collections of user-saved or application-supplied or driver-supplied settings for the Print dialog). The Print dialog will allow a user to choose from existing settings or to create new saved settings.

3. Dependencies between user options. User options are described in "Printing Dialog Extensions and User Options" (page 13). Don't implement your own mechanism for dependencies between user options.

Preliminary © Apple Computer, Inc. 5/23/00

Overview

The Mac OS X Printing Manager is part of a robust new printing system in Mac OS X that provides a refined user interface, as well as many sophisticated features for developers who work with printing. For a detailed overview of the printing system, see the "System Architecture" chapter of "Inside Mac OS X: System Overview."

All Carbon applications must convert their printing code to use the Carbon Printing Manager API, which provides access to many of the printing features available in Mac OS X. The Carbon Printing Manager allows Carbon applications to print in Mac OS 8 and Mac OS 9 with existing printer drivers and in Mac OS X with its new printing architecture.

The Carbon Printing Manager also provides a printing session mechanism, where a session encapsulates one entire printing cycle—that is, the steps from bringing up the Print dialog to printing the document. In Mac OS 8 and 9, only one printing session is allowed at a time, but in Mac OS X the session API supports multiple, simultaneous printing sessions. The Carbon Printing Manager also supports sheets, a per-document mechanism for displaying Print or Page Setup information.

The Mac OS X Printing Manager supports additional printing features, including a powerful mechanism for extending printing dialogs—the printing dialog extension (PDE) mechanism documented here.

The following sections provide a brief description of the traditional mechanism for extending printing dialogs, outline the new PDE mechanism, and offer some advice on when to use the new PDE mechanism:

- "Extending Printing Dialogs in Classic Mac OS and in Carbon" (page 12)
- "Extending Printing Dialogs in Mac OS X" (page 13)
- "When You Should Use Printing Dialog Extensions" (page 18)

Preliminary © Apple Computer, Inc. 5/23/00

Extending Printing Dialogs in Classic Mac OS and in Carbon

An application can extend a printing dialog in Classic Mac OS using the mechanism described in the "Printing Manager" chapter of "Inside Macintosh: Imaging With QuickDraw" and updated in "Technote 1080: Adding Items to the Printing Manager's Dialogs." That mechanism, sometimes referred to as the "AppendDITL" approach, requires the application to provide routines to initialize the Print or Page Setup dialog, append new items to it, draw the items, handle user actions on the appended items, and so on.

The AppendDITL mechanism is supported by the Carbon Printing Manager. Carbon applications that use this mechanism to modify printing dialogs will work both in Mac OS 8 and 9 and in Mac OS X. The Carbon Printing Manager API provides access to the fields of the printing dialog. However, Carbon does not permit direct access to `TPrDlg`, as described in Technote 1080, so applications that use AppendDITL will require some modification.

Using the AppendDITL mechanism can provide the following advantages:

- If you've already written code to extend printing dialogs, you can reuse it.
- Your Carbon application can use the same code when running in Mac OS 8 and 9 and in Mac OS X.

However, there are significant drawbacks to using the AppendDITL mechanism to extend printing dialogs:

- You can't use sheets, a per-document mechanism for displaying Print or Page Setup information in Mac OS X.
- You can't extend printing dialogs in Mac OS X with the full flexibility available from printing dialog extensions (PDEs), as described throughout this document.

  For example, you can add only one panel per dialog, with a limited amount of screen space for your dialog additions.
- When a user saves a collection of settings from a Print dialog in Mac OS X, you won't be able to easily save your extended information with the settings.
- You won't be able to provide a summary of user option information to the user.

Preliminary © Apple Computer, Inc. 5/23/00

Extending Printing Dialogs in Mac OS X

Applications running in Mac OS X can use the Mac OS X Printing Manager's printing dialog extension (PDE) mechanism to modify and extend the Page Setup and Print dialogs, which are shown in Figure 1-1 (page 15) andFigure 1-3 (page 23), respectively.

The following sections provide an overview of working with PDEs in Mac OS X:

Printing Dialog Extensions and User Options

A **user option** is a collection of related user interface elements which appear together in a panel of a printing dialog. For example, Figure 1-3 (page 23) shows the "Copies & Pages" user option.

A **printing dialog extension (PDE)** is a code module, implemented as a Core Foundation plug-in, that implements one or more user options. For more detail on how a PDE is constructed, see "Printing Dialog Extension Architecture" (page 19).

A PDE typically implements its user interface with an embedded control hierarchy and uses Carbon event handlers to respond to user interaction with the option's controls. You can find information on designing controls in Aqua in the document "Adopting the Aqua Interface." Additional user interface information on printing dialogs will be provided in a future, comprehensive document on the Aqua interface. For related information on controls, see "Controls in Printing Dialog Extensions" (page 17).

A **user option kind ID** is a string that identifies a user option independently of its title. If an application implements PDEs for more than one kind of user option, it can identify the type of a PDE by its kind ID. For information on user option kind IDs defined by the Mac OS X Printing Manager, see "Universal and Standard User Option Kind ID Constants" (page 79).

Preliminary © Apple Computer, Inc. 5/23/00

Kinds of User Options

Your application can supply PDEs to override universal user options or to implement standard and custom options.

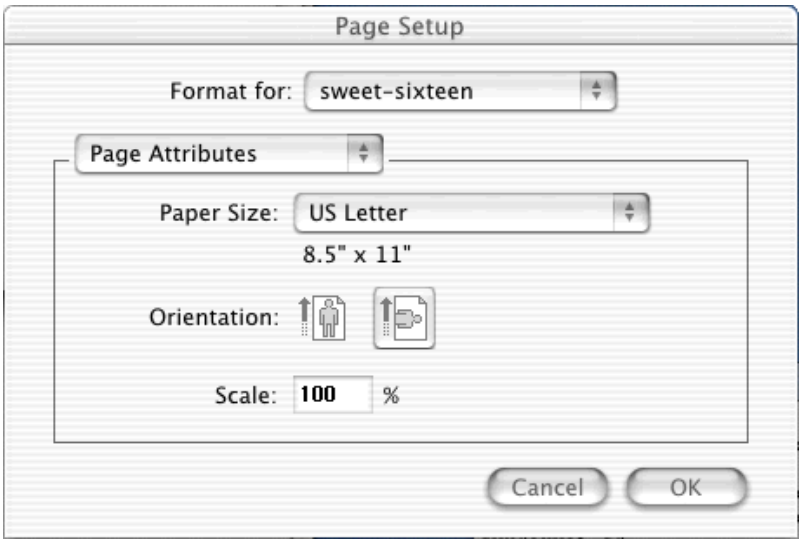
A **universal user option** is implemented as part of the Mac OS X Printing Manager and thus always available. The Mac OS X Printing Manager defines kind IDs and titles for universal user options, supplies PDEs that implement the options, and is responsible for providing localized versions of all universal title strings. For an example of a universal user option, see Figure 1-3 (page 23), which shows the "Copies & Pages" option on the Print dialog.

A **standard user option** has a standard user kind ID and title defined by the Mac OS X Printing Manager, but is not implemented by the Printing Manager. Standard user options help provide users with a consistent printing experience when switching between different applications and printers. "Color" and "Paper Source" are examples of standard user options. The Mac OS X Printing Manager is responsible for providing localized versions of all standard title strings.

A **custom user option** is unique to an application or destination printer. Applications that implement custom user options define a kind ID for each option and are responsible for localization of all text within the option, including its title. The Mac OS X Printing Manager does not modify the appearance of custom user options in any way.

Preliminary © Apple Computer, Inc. 5/23/00

Figure 1-1 A Page Setup dialog showing a universal user option (Page Attributes)



Working With Universal and Standard Options

Your application can supply PDEs to override universal options and to implement standard user options. In both cases, any title string supplied by the PDE is ignored; the string defined by the Mac OS X Printing Manager is always used.

A PDE that overrides a universal user option must provide all the original functionality of that user option, as well as any added functionality. Specifically, the overriding PDE must supply all the public data defined for the overridden option in the Page Format or Print Settings ticket, using the keys and data types defined by the Mac OS X Printing Manager. The PDE may store any additional private data in the ticket using its own tags. For related information, see "Defining Constants for the Print Settings Ticket" (page 24).

For standard user options, the Printing Manager defines only the title and user option kind ID, not the data format. A PDE that implements a standard user option is free to store the settings for that user option in any format and with any tags it wishes in the Page Format or Print Settings ticket, as appropriate. However, it is

Preliminary © Apple Computer, Inc. 5/23/00

recommended that the PDE use types defined by Core Foundation to store its information in a ticket, so that the ticket can be flattened and unflattened (that is, converted to XML and written to disk, or read from disk and translated from XML).

If a PDE overrides a user option, it must provide an implementation. Once a user option is registered, a panel is reserved for it and it is expected that the PDE will display the user option within its assigned panel in accordance with the PDE API.

The Printing Manager uses the following order of precedence to determine which user option is displayed in the printing dialog if more than one PDE attempts to register the same user option kind:

1. Application
2. Printer Module
3. Printing Manager

An application's user option is displayed in preference to any other option with the same kind. A printer module's user option is displayed in preference to a user option with the same kind implemented by the Printing Manager. A Printing Manager user option appears only if no other PDE registers an option with the same kind.

The Printing Manager registers user options according to its order of precedence: application user options, followed by printer module user options, followed by Printing Manager user options. Once an application registers a universal or standard user option, a printer module or Printing Manager PDE will not be able to register that user option. Similarly, a Printing Manager PDE cannot register a PDE that has already been registered by an application or printer module.

If, for any reason, a user option is overridden during the execution of the dialog (because, for example, the user switches printer modules), the `Terminate` (page 69) function of the overridden PDE is called and the controls of the overridden user option are deleted.

Because a PDE that overrides a universal user option completely replaces that option, the overriding PDE is responsible for providing all the basic functionality of the overridden option. It must also respond to any Mac OS X Printing Manager APIs which affect the universal user option (such as setting the number of copies). When a Mac OS X Printing Manager function that could affect an overriding user option is called, the Mac OS X Printing Manager calls the PDE's `Sync` (page 65)

Preliminary © Apple Computer, Inc. 5/23/00

function with the `reinitializePlugIn` parameter set to `TRUE`. The PDE should then examine the page format and print settings objects and adjust its controls accordingly.

Controls in Printing Dialog Extensions

The hosting of human interface elements in Mac OS X Printing Manager dialogs is based on the Carbon Control Manager. The Carbon Control Manager supports embedding controls within user panes, which allows the owner of the user pane to manipulate those embedded controls as a group, without knowing precisely which controls have been embedded.

When the Printing Manager needs to draw the human interface elements of a printing dialog extension, it simply makes the user pane in which they are embedded visible. The Control Manager then draws the embedded controls. Updates and activate/deactivate events are handled through the Control and Window Managers.

If a human interface element requires explicit handling by the printing dialog extension, it should register a drawing callback with the Control Manager when the element is created in the PDE's `Initialize` (page 63) function.

All event handling is performed through the Carbon Control Manager's event handling model. A user interface element registers an event handler, which is called whenever a user interacts with the element. The embedding hierarchy ensures that the proper handlers are called.

Advantages and Disadvantages of PDEs

Using printing dialog extensions (PDEs) to extend printing dialogs in Mac OS X can provide many advantages:

- You can extend a printing dialog with as many additional user option panels as necessary.
- A printing dialog extended with a PDE can be displayed as a per-document sheet in Mac OS X (although supporting sheets requires some additional work).
- You can override the universal user options supplied by the Mac OS X Printing Manager (such as the "Copies & Pages" option).
- You can ask for an increased panel size for a user option.

Preliminary © Apple Computer, Inc. 5/23/00

- When a user saves a collection of settings from a Print dialog in Mac OS X, you can easily save information from your user options with the settings.
- You can provide summary information about user options to the user in the "Summary" pane.

There are also some potential disadvantages to using the PDE mechanism in your Carbon application:

- A PDE can't directly reuse "AppendDITL" code you may have already written.
- A PDE can only extend printing dialogs for an application running in Mac OS X. Your application must provide additional code to extend printing dialogs in Mac OS 8 and 9.

When You Should Use Printing Dialog Extensions

Based on the previous sections, we can make the following statements about when to use printing dialog extensions (PDEs) and when to use the traditional AppendDITL mechanism:

- If you want to write (or reuse) one chunk of code that extends printing dialogs in Mac OS 8 or 9 and Mac OS X, you can use the traditional AppendDITL approach, though you won't be able to take advantage of some advanced printing features available in Mac OS X.
- If you need to extend printing dialogs in Mac OS 8 or 9 and Mac OS X and want to take advantage of the advanced Mac OS X features available through PDEs, your application should include code for both the AppendDITL approach (for Mac OS 8 and 9) and the PDE approach, and determine at run time which to use.
- If your application is going to run only in Mac OS X, keep reading—it's time to learn how to write PDEs for the Mac OS X Printing Manager.

Preliminary © Apple Computer, Inc. 5/23/00

Printing Dialog Extension Architecture

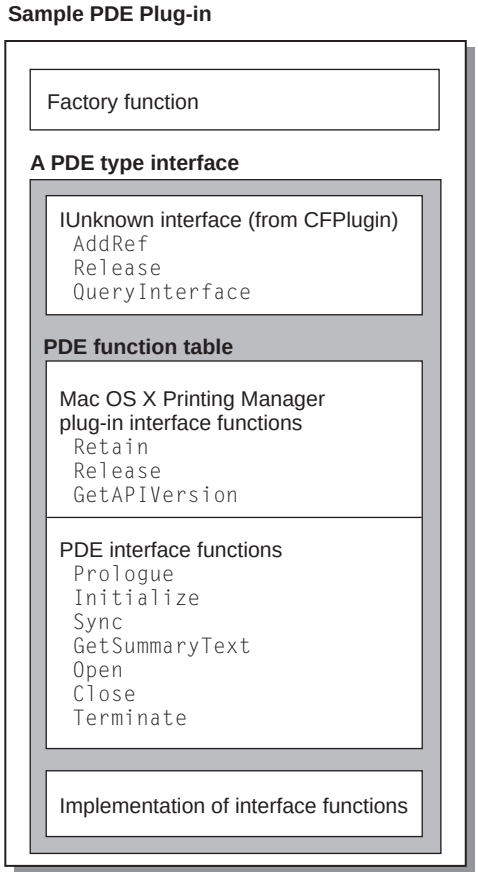
To understand the architecture of printing dialog extensions, you should be familiar with the information in the document "Core Foundation Plug-In Services." This section provides a brief overview of the relationship between printing dialog extensions and the Core Foundation plug-in model.

A sample PDE is shown in Figure 1-2. A PDE is a Core Foundation plug-in and includes the following, all implemented by the plug-in developer:

- A factory function, which knows how to create an instance of the IUnknown interface. The IUnknown interface is described in "Core Foundation Plug-in Services."
- An implementation of the IUnknown interface, based on the `IUnknownVTbl` structure defined in CFPlugIn.h.
  - The `QueryInterface` function knows how to create an instance of a PDE function table.
- An implementation of the functions defined in the PDE function table. The PDE function table is based on two structures:
  - The `PMPlugInHeaderVTable` (page 72) structure defined in PMPluginHeader.h. All Mac OS X Printing Manager plug-ins must implement these functions.
  - The `PlugInIntfVTable` (page 73) structure defined in PMPrintingDialogExtensions.h. All PDEs must implement these functions.

Preliminary © Apple Computer, Inc. 5/23/00

Figure 1-2 A PDE plug-in, showing the main interface functions



The Mac OS X Printing Manager depends on the following process to load your PDE and to create instances of the PDE interface (or function table) to display your user option in a Print or Page Setup dialog:

1. Before your application calls the `PMSessionPrintDialog` function or the `PMSessionPageSetupDialog` function to display a printing dialog, it must call the CFPlugIn function `CFPlugInCreate` to create an instance of the PDE. This function registers information about your PDE, including the types and interfaces it supports and a factory function for each type.

Preliminary © Apple Computer, Inc. 5/23/00

2. Before displaying a printing dialog, the Mac OS X Printing Manager uses CFPlugIn to obtain a reference to a function table for each registered PDE plug-in type.
    a. The factory function creates an instance of the IUnknown interface for the PDE.
    b. The `QueryInterface` function of the IUnknown interface creates an instance of the PDE interface (or function table).
3. For each such PDE function table, the Printing Manager calls the `Prologue` (page 61) function to get information about the user option, then the `Initialize` (page 63) function, to set up the user option.

To support this process, you perform the tasks described in the next section, "Creating a Printing Dialog Extension" (page 21).

Okay, let's get to work on that PDE.

Creating a Printing Dialog Extension

This section shows in detail how to create a custom application printing dialog extension (PDE). The sample code implements a simple custom user option called "Sample Application PDE" for the Print dialog. The user option displays a "Print Selected Text Only" checkbox.

Before starting on your PDE, you should have read the section "Printing Dialog Extension Architecture" (page 19).

To create a PDE and use it in your application, you complete the tasks described in the following sections:

1. "Designing a User Interface" (page 22)
2. "Defining Factory and Type IDs" (page 23)
3. "Defining a User Option Kind ID" (page 24)
4. "Defining Constants for the Print Settings Ticket" (page 24)
5. "Defining Additional Data Structures" (page 25)

Preliminary © Apple Computer, Inc. 5/23/00

6. "Writing Code to Implement Your PDE" (page 29)
7. "Creating a Custom Information Property List" (page 53)
8. "Using a PDE in Your Application" (page 55)

Designing a User Interface

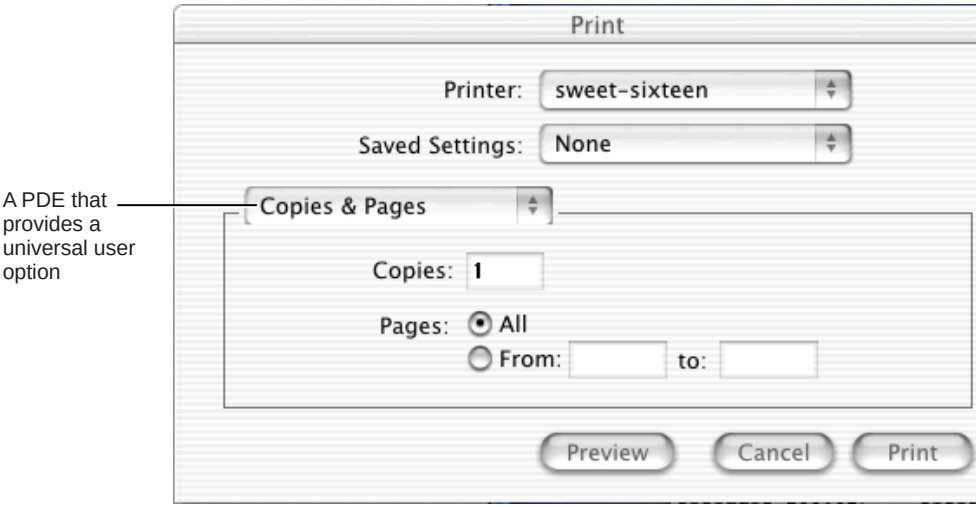
A printing dialog extension implements one or more user options—collections of related user interface elements which appear together in a panel of a printing dialog. Your application can provide user options for both the Page Setup and Print dialogs. A user option may consist of multiple controls, but these controls should generally refer to a single feature—for example, the "Copies & Pages" option in the Print dialog shown in Figure 1-3 (page 23).

There are no restrictions on a PDE's ability to display dialogs, alerts, and other elements. You are encouraged, however, to avoid excessive use of additional windows and dialogs, which can clutter the Mac OS X Printing Manager human interface and degrade the user experience.

Consult the document "Adopting the Aqua Interface" for information on the size and positioning of controls in Aqua.

Preliminary © Apple Computer, Inc. 5/23/00

Figure 1-3 A Print dialog, showing a universal user option (Copies & Pages)



The sample code in this document implements a PDE that adds a simple custom user option to the Print dialog to specify printing the current selection. The PDE title is "Sample Application PDE". It is up to your PDE to supply a localized version of the title string, which appears in the settings panels navigation menu (the pop-up menu that shows the user option "Copies & Pages" in Figure 1-3). For information on localization, see the document "Core Foundation Bundle Services."

Defining Factory and Type IDs

Because printing dialog extensions are based on the CFPlugIn model, they must supply at least one factory ID to associate with the CFPlugInFactories key and at least one type ID to associate with the CFPlugInTypes key.

The Mac OS X Printing Manager defines type IDs for PDEs. These type IDs are described in "Printing Dialog Extension Type ID Constants" (page 77). You only need to define a factory ID for your PDE's factory function. The sample application PDE defines the following constant for a factory ID:

Preliminary © Apple Computer, Inc. 5/23/00

Listing 1-1 A sample factory ID

```
#define kSampleAppPDEIntfFactoryIDStr "BED2EC92-E57F-11D3-80C9-0050E4603277"
```

The sample PDE then uses this constant in the `SampleAppPDEPluginFactory` (page 30) function to create a UUID (Universally Unique Identifier) for the factory and to store it in the instance of the IUnknown interface the factory creates. UUIDs are described in the document "Core Foundation Utility Services."

A standard way to create associations between CFPlugIn keys and PDE constants is to store the information in a custom information property list. For information on how to do so, see "Creating a Custom Information Property List" (page 53).

Defining a User Option Kind ID

If you are writing a custom PDE, you need to define a string constant that specifies the user option kind. The sample application PDE defines the following constant for that purpose:

Listing 1-2 A sample user option kind ID

```
#define kSampleAppUserOptionKindID    "com.appvendor.print.pde.PrintSelectionOnly"
```

The user option kind ID should be unique, and a string like the one shown here, where you replace `appvendor` with your information, serves that purpose. The sample application PDE uses this constant in its `Prologue` (page 61) function, which is shown in Listing 1-14 (page 39).

If you are overriding a universal option or implementing a standard option, you use one of the kind IDs defined in "Universal and Standard User Option Kind ID Constants" (page 79).

Defining Constants for the Print Settings Ticket

Print job information associated with printing dialogs is stored in objects referred to as tickets—either Page Format tickets or Print Settings tickets. To store information for a Print dialog user option in a Print Settings ticket and retrieve it from the ticket, you must define a key constant to identify the data in the ticket. (You would define similar constants to work with information for a Page Setup user option in a Page

Preliminary © Apple Computer, Inc. 5/23/00

Format ticket.) The sample application PDE uses the following constants to specify a key for the one data type it needs to store and retrieve (the state of the checkbox that indicates whether to print the current selection only):

Listing 1-3 Sample constants for storing and retrieving Print Settings ticket data

```
#define kAppPrintSelectionOnlyStr     kPMPrintSettingsPrelude "AppPrintSelectionOnly"
#define kAppPrintSelectionOnlyKey     CFSTR( kAppPrintSelectionOnlyStr )
```

`kAppPrintSelectionOnlyStr`

Concatenates a string identifying our user option with a prelude string to make a unique identifier for the option data. The constant `kPMPrintSettingsPrelude` is defined by the Mac OS X Printing Manager. You should define your own prelude string for your PDE.

`kAppPrintSelectionOnlyKey`

Creates a `CFSTR` string, based on the string defined by `kAppPrintSelectionOnlyStr`, that can be used as a key to store information in or extract information from a Print Settings ticket.

Listing 1-16 (page 42) shows how the sample application PDE uses the key `kAppPrintSelectionOnlyKey` in its `MyPDESync` function to extract information from a Print Settings ticket or to add it to a ticket.

Defining Additional Data Structures

You will need to define certain additional data structures for your PDE. For example, the PDE "Sample Application PDE" defines a structure for the IUnknown interface, a structure for its own interface, and a structure to store its private context. These structures are described in the following sections:

"Defining an Unknown Interface Structure" (page 26)

"Defining a PDE Interface Structure" (page 27)

"Defining a Private Context" (page 28)

Preliminary © Apple Computer, Inc. 5/23/00

Defining an Unknown Interface Structure

Listing 1-5 (page 27) shows `IUnknownInstance`, the interface structure for the IUnknown interface that is defined by the sample application PDE. All PDEs must define an interface structure similar to this one—starting with the CFPlugIn data type `IUnknownVTbl`— and provide a factory function that knows how to create an instance of the interface. For information on the IUnknown interface functions, see "Implementing Functions for the CFPlugIn Interface" (page 31). For an overview of the entire PDE plug-in, see "Printing Dialog Extension Architecture" (page 19).

The `SampleAppPDEPluginFactory` function, shown in Listing 1-7 (page 30), creates an instance the IUnknown interface for the sample application PDE.

Listing 1-4 A sample structure for an unknown interface

```
typedef struct
{
    // Pointer to the vtable (defined in CFPlugIn.h):
    const IUnknownVTbl*      vtable;

    // Our vtable storage:
    IUnknownVTbl             vtableStorage;

    // Factory ID this instance is for:
    CFUUIDRef                factoryID;

    // Reference counter:
    ULONG                    refCount;

}
IUnknownInstance;
```

`vtable`

A pointer to a vtable, or table of interface callback functions. Points to the `vtableStorage` field.

`vtableStorage`

The actual `IUnknownVTbl` structure, defined in CFPlugIn.h. This table includes entries for the callback functions `AddRef`, `Release`, and `QueryInterface`. The versions of these functions implemented by the

Preliminary © Apple Computer, Inc. 5/23/00

sample application PDE, `MyCFAddRef`, `MyCFRelease`, and `MyCFQueryInterface`, are shown in Listing 1-8 (page 32), Listing 1-9 (page 33), and Listing 1-10 (page 34), respectively. The `MyCFQueryInterface` function is also described in "Printing Dialog Extension Architecture" (page 19).

`factoryID`

Stores an ID that identifies the factory function that can create instances of this interface. For related information, see "Defining Factory and Type IDs" (page 23).

`refCount`

A counter used by the functions `AddRef` and `Release` to keep track of references to an instance of the interface.

The sample application PDE described in this document implements separate reference counting for the IUnknown interface defined here and the PDE interface defined in `PMPlugInHeaderVTable` (page 72). However, your PDE may wish to combine its reference counting.

Defining a PDE Interface Structure

Listing 1-5 (page 27) shows the interface structure for the sample printing dialog extension "Sample Application PDE," which implements a user option to print only the current selection. An instance of this interface provides the function table for the PDE interface, as well as a reference count variable for the instance. For an overview of the PDE interface, see "Printing Dialog Extension Architecture" (page 19).

All PDEs must implement an interface similar to this one and provide a function similar to the `MyCFQueryInterface` (page 34) function that knows how to create an instance of the interface. However, your PDE does not necessarily need to define its own PDE interface structure. You can add use the `PlugInIntf` (page 73) structure defined in PMPrintingDialogExtensions.h and add functionality to the PDE using a private context structure. For more information, see "Defining a Private Context" (page 28).

Listing 1-5 A sample structure for a PDE interface

```
typedef struct
{
    PlugInIntfVTable     *vtable;
```

```
    UInt32              refCount;
}
PrintSelOnlyPlugInInterface;
```

`vtable`

A pointer to a table of callback functions. The interface for all Mac OS X Printing Manager plug-ins, including PDEs, must start with the header interface defined by the `PMPlugInHeaderVTable` (page 72) structure. In the interface defined here, the `vtable` field is a pointer to a `PlugInIntfVTable` (page 73) structure, which in turn has a structure of type `PMPlugInHeaderVTable` as its first element.

`refCount`

The sample application PDE uses this field to keep track of references to an instance of the plug-in. To see how this is done in the sample application PDE, see the `MyPMRetain` function, shown in Listing 1-11 (page 36), and the `MyPMRelease` function, shown in Listing 1-12 (page 37).

Defining a Private Context

In Mac OS X, a user can have multiple, simultaneous print sessions—for example, a user can display sheets for printing two documents in the same application at the same time. The Mac OS X Printing Manager calls on the PDE plug-in to create a new instance of its interface (or function table) for each session. Each instance accesses the same plug-in code, but can contain its own private context, which allows the plug-in to be reentrant.

After the Mac OS X Printing Manager obtains a printing dialog extension (PDE) function table, it calls the PDE's `Prologue` (page 61) function to obtain information about the PDE's user option. The `Prologue` function may also provide a private context. The Printing Manager passes this context in subsequent calls to other PDE functions.

You can use the private context to provide your PDE with access to data or functions. The sample application PDE defines the context structure shown in Listing 1-6. This private context stores the location (in local coordinates) and dimensions of the user option's drawing area, as well as a reference to the option's lone control, a checkbox. The sample PDE's `MyPDEPrologue` function, shown in Listing 1-14 (page 39), calls `InitContext` (page 51) to create and initialize the context.

Preliminary © Apple Computer, Inc. 5/23/00

`InitContext` merely initializes the context to safe values—the context gets fully initialized when the Printing Manager calls the `MyPDEInitialize` (page 40) function, which in turn calls the `MyPDESync` (page 42) function.

Listing 1-6 A sample structure for a private PDE context

```
typedef struct
{
    Rect        theFrameRect;    // This plug-in's drawing area.
    ControlRef  thePrintSelTextOnlyControlRef;
}
PrintSelectionOnlyContext;
```

Writing Code to Implement Your PDE

"Printing Dialog Extension Architecture" (page 19) describes the basic layout of a printing dialog extension plug-in. To implement a PDE, you need to implement the following functions:

- A factory function that knows how to create an instance of the IUnknown interface.
- The IUnknown interface functions from CFPlugIn, which are defined by the `IUnknownVTbl` structure in CFPlugIn.h. For more information, see the document "Core Foundation Plug-in Services."
- The functions that are required in all Mac OS X Printing Manager plug-ins. These are defined in the `PMPlugInHeaderVTable` (page 72) structure in PMPluginHeader.h.
- The functions specific to the PDE interface. These are defined in the `PlugInIntfVTable` (page 73) structure, which includes a `PMPlugInHeaderVTable` structure as its first element. These structures are defined in PMPrintingDialogExtensions.h.
- Any additional functions required by your PDE.

The following sections describe how to implement the required functions for a sample application PDE and include sample code for each of the functions:

- "Implementing a Factory Function" (page 30)

Preliminary © Apple Computer, Inc. 5/23/00

Implementing a Factory Function

Mac OS X Printing Manager obtains an instance of the IUnknown interface, created by the plug-in's factory function, for each registered plug-in. The sample application PDE implements the following factory function:

SampleAppPDEPluginFactory

The `SampleAppPDEPluginFactory` function is the factory function for the sample application PDE. It creates an unknown instance of type `IUnknownInstance` and initializes its fields.

Listing 1-7 A sample plug-in factory function

```
void* SampleAppPDEPluginFactory( CFAllocatorRef allocator, CFUUIDRef
reqTypeID )
{
    CFUUIDRef            myIntfID;
    CFUUIDRef            myFactoryID;
    OSStatus             err = noErr;
    IUnknownInstance*    instance = NULL;

    // There is not much we can do with errors - just return NULL.
    myIntfID = CFUUIDCreateFromString( CFAllocatorGetDefault(),
                    CFSTR(kAppPrintDialogTypeIDStr) );

    // If the requested type matches our plug-in type (it should!)
    // have a plug-in instance created which will query us for
    // interfaces:

    if( CFEqual( reqTypeID, myIntfID ))
    {
        instance = (IUnknownInstance*) malloc( sizeof( IUnknownInstance ));
```


Preliminary © Apple Computer, Inc. 5/23/00

```
        if (instance != NULL)
        {
            // Clear all object memory:
            memset( instance, 0, sizeof( IUnknownInstance ));

            // Assign all members:
            instance->vtable = &instance->vtableStorage;

            instance->vtableStorage.QueryInterface = MyCFQueryInterface;
            instance->vtableStorage.AddRef = MyCFAddRef;
            instance->vtableStorage.Release = MyCFRelease;

            myFactoryID = CFUUIDCreateFromString( CFAllocatorGetDefault(),
                            CFSTR(kSampleAppPDEIntfFactoryIDStr) );

            instance->factoryID = myFactoryID;
            instance->refCount = 1;

            // Register the newly created instance
            // for our factory with Core Foundation:
            CFPlugInAddInstanceForFactory( myFactoryID );
        }
    }
    CFRelease( myIntfID );

    return ( (void*) instance );
}
```

Implementing Functions for the CFPlugIn Interface

The IUnknown interface, based on the `IUnknownVTbl` structure defined in CFPlugIn.h, is a required part of all plug-ins based on CFPlugIn, including all Mac OS X Printing Manager plug-ins. The `IUnknownInstance` structure, which is shown in Listing 1-4 (page 26), is the version of the IUnknown interface defined by the sample application PDE and created by the `SampleAppPDEPluginFactory` (page 30) function.

The following sections show how the sample PDE implements the functions of the `IUnknownInstance` interface:

- `MyCFAddRef` (page 32)

Preliminary © Apple Computer, Inc. 5/23/00

- MyCFRelease (page 32)
- MyCFQueryInterface (page 34)

MyCFAddRef

The MyCFAddRef function works with MyCFRelease (page 32) to track references to an instance of a plug-in and free the instance when it is no longer referenced. It merely casts the passed object to its own type, IUnknownInstance, increments the reference count, and returns the new count.

Listing 1-8 A sample IUnknown interface AddRef function

```
static ULONG MyCFAddRef( void *obj )
{
    IUnknownInstance *instance = (IUnknownInstance*) obj;
    ULONG    refCount = 0;

    // We can't do much with errors here since we can only return
    // updated reference count value.

    if( NULL != instance )
    {
        // Get updated refCount value (should be under mutex):
        refCount = ++instance->refCount;
    }
    else
    {
        refCount = 0;
    }
    return ( refCount );
}
```

MyCFRelease

The MyCFRelease function works with MyCFAddRef (page 32) to track references to an instance of a plug-in and free the instance when it is no longer referenced. MyCFRelease casts the passed object to its own type, IUnknownInstance, decrements the reference count, and if the count reaches 0, unregisters the plug-in with the OS and frees the plug-in instance.

Preliminary © Apple Computer, Inc. 5/23/00

Listing 1-9 A sample IUnknown interface Release function

```
static ULONG MyCFRelease( void *obj )
{
    IUnknownInstance*    instance = (IUnknownInstance*) obj;
    ULONG                refCount = 0;

    // We can't do much with errors here since we can only return
    // updated reference count value.
    try
    {
        if( instance == NULL )
        {
            // Get updated refCount value (should be under mutex):

            // Make sure refCount is non-zero:
            if( instance->refCount == 0 )
            {
                instance = NULL;
                return(refCount);
            }
            refCount = --instance->refCount;

            // Is it time to self-destruct?
            if( refCount == 0 )
            {
                // Unregister 'instance for factory' with Core Foundation:
                CFPlugInRemoveInstanceForFactory( instance->factoryID );

                // Release used factoryID:
                CFRelease( instance->factoryID );
                instance->factoryID = NULL;

                // Deallocate object's memory block:
                free( (void*) instance );
                instance = NULL;
            }
        }
    }
    catch( ... )
    {
```


Preliminary © Apple Computer, Inc. 5/23/00

```
        if( instance )
        {
            // Release used factoryID:
            if( instance->factoryID )
                CFRelease( instance->factoryID );

            // Deallocate object's memory block:
            free( (void*) instance );
        }
        // Return zero refCount value:
        refCount = 0;
    }
    return refCount;
}
```

MyCFQueryInterface

As described in "Printing Dialog Extension Architecture" (page 19), the Mac OS X Printing Manager calls the IUnknown instance's `QueryInterface` function to create an instance of the PDE. The `MyCFQueryInterface` function shown here calls the `CreatePlugInInterface` (page 49) function to return the actual PDE instance.

Listing 1-10 A sample IUnknown interface QueryInterface function

```
static HRESULT MyCFQueryInterface( void *obj, REFIID iID, LPVOID *intfPtr )
{
    IUnknownInstance*    instance = (IUnknownInstance*) obj;
    CFUUIDRef            myIntfID = NULL, reqIntfID = NULL;
    HRESULT              err = E_UNEXPECTED;
    PlugInIntf*          interface;

    // Get IDs for requested and PDE interfaces:
    reqIntfID = CFUUIDCreateFromUUIDBytes( CFAllocatorGetDefault(), iID );
    myIntfID = CFUUIDCreateFromString( CFAllocatorGetDefault(),
                    CFSTR(kDialogExtensionIntfIDStr) );

    // If we are asked to return the interface for the
    // IUnknown vtable, which the system already has access to,
    // just increment the refcount value
```


Preliminary © Apple Computer, Inc. 5/23/00

```
    if( CFEqual( reqIntfID, IUnknownUUID ) )
    {
        instance->vtable->MyCFAddRef( (void*) instance );
        *intfPtr = (LPVOID) instance;
        err = S_OK;
    }
    else    // if we are asked for the PDEs interface,
            // make one and return it.
    if ( CFEqual( reqIntfID, myIntfID ) )
    {
        err = CreatePlugInInterface( &interface );
        if ( err == noErr )
        {
            *intfPtr = (LPVOID) interface;
            err = S_OK;
        }
    }
    else // we will return the err = E_NOINTERFACE and a  *intfPtr of NULL;
    {
        *intfPtr = NULL;
        err = E_NOINTERFACE;
    }
    // Clean up and return status:
    CFRelease( reqIntfID );
    CFRelease( myIntfID );

    return( err );
}
```

Implementing Functions Required by All Printing Plug-ins

All printing plug-ins in the Mac OS X printing system, including printing dialog extensions, must implement the functions defined in the `PMPlugInHeaderVTable` (page 72) structure and described in the following sections:

- `MyPMRetain` (page 36)
- `MyPMRelease` (page 36)
- `MyPMGetAPIVersion` (page 37)

Preliminary © Apple Computer, Inc. 5/23/00

MyPMRetain

`Retain` — Increments the reference count for an instance of a plug-in. Your retain function increments the count of references to your PDE. (page 58)

The previous paragraph describes a generic `Retain` function. The version below, implemented by the sample application PDE, casts the passed object pointer of type `PMPlugInHeaderInterface` (page 72) to its own `PrintSelOnlyPlugInInterface` type (shown in Listing 1-5 (page 27)) to get access to its reference count. This cast works because both types start with a `PMPlugInHeaderVTable` (page 72) reference.

Listing 1-11 A sample Printing Manager plug-in Retain function

```
static OSStatus MyPMRetain( PMPlugInHeaderInterface *obj )
{
    if (obj != NULL)
    {
        PrintSelOnlyPlugInInterface *plugin =
                        (PrintSelOnlyPlugInInterface*) obj;

        // Increment reference count:
        plugin->refCount++;
    }
    return noErr;
}
```

MyPMRelease

`Release` — Decrements the reference count for an instance of a plug-in. Your release function sets the passed object pointer to NULL, decrements the reference count for your PDE, and if the count reaches 0, frees the PDE and any related storage. (page 59)

The previous paragraph describes a generic `Release` function. The version below, implemented by the sample application PDE, casts the passed object pointer of type `PMPlugInHeaderInterface` (page 72) to its own `PrintSelOnlyPlugInInterface` type (shown in Listing 1-5 (page 27)) to get access to its reference count. This cast works because the definition of each type starts with a `PMPlugInHeaderVTable` (page 72) reference. If the reference count goes to 0, `MyPMRelease` disposes of the PDE's function table, then of the PDE instance itself.

Preliminary © Apple Computer, Inc. 5/23/00

Listing 1-12 A sample Printing Manager plug-in Release function

```
static OSStatus MyPMRelease( PMPlugInHeaderInterface **objPtr )
{
    if (*objPtr != NULL)
    {
        PrintSelOnlyPlugInInterface* plugin =
                            (PrintSelOnlyPlugInInterface*) *objPtr;

        // Clear caller's variable:
        *objPtr = NULL;

        // Decrement reference count:
        plugin->refCount--;

        // When reference count is one it's time to self-destruct:
        if( plugin->refCount == 0 )
        {
            // Delete object's vtable:
            DisposePtr( (char *)plugin->vtable );

            // Delete object's memory block:
            DisposePtr( (char *)plugin );
        }
    }
    return noErr;
}
```

MyPMGetAPIVersion

`GetAPIVersion` — Supplies API version information for a plug-in. Your version function supplies information that can be used to determine PDE compatibility. (page 59)

The previous paragraph describes a generic `GetAPIVersion` function. The version below, implemented by the sample application PDE, simply sets the fields of the version structure pointed to by the passed version pointer to constant values defined in "Printing Dialog Extension Interface Version Constants" (page 77).

Preliminary © Apple Computer, Inc. 5/23/00

Listing 1-13 A sample Printing Manager plug-in GetAPIVersion function

```
static OSStatus
MyPMGetAPIVersion(
            PMPlugInHeaderInterface *obj, PlugInAPIVersion *versionPtr )
{
    // Return versioning info:
    versionPtr->buildVersionMajor = kPDEBuildVersionMajor;
    versionPtr->buildVersionMinor = kPDEBuildVersionMinor;
    versionPtr->baseVersionMajor  = kPDEBaseVersionMajor;
    versionPtr->baseVersionMinor  = kPDEBaseVersionMinor;

    return noErr;
}
```

Implementing the PDE Interface

All printing dialog extension plug-ins must implement the functions defined in the `PlugInIntfVTable` (page 73) structure and described in "Printing Dialog Extension Callback Functions" (page 60). The following sections provide sample implementations of these functions:

- `MyPDEPrologue` (page 38)
- `MyPDEInitialize` (page 40)
- `MyPDESync` (page 42)
- `MyPDEGetSummaryText` (page 44)
- `MyPDEOpen` (page 47)
- `MyPDEClose` (page 47)
- `MyPDETerminate` (page 48)

MyPDEPrologue

`Prologue` — Called by the Mac OS X Printing Manager after it obtains a printing dialog extension (PDE) plug-in interface (or function table). Your prologue function returns information about the PDE's user option and may also provide its own private context information. (page 61)

Preliminary © Apple Computer, Inc. 5/23/00

The previous paragraph describes a generic `Prologue` function. The version below, implemented by the sample application PDE, does in fact provide context information, which it gets by calling the `InitContext` (page 51) function. It also provides the other information needed by the Mac OS X Printing Manager, including user option title, screen space needed by the user panel, and so on.

Listing 1-14 A sample PDE Prologue function

```
static OSStatus
MyPDEPrologue(PMPDEContext*context,
              OSType      *creator,
              CFStringRef  *userOptionKind,
              CFStringRef  *title,
              UInt32      *maxH,
              UInt32      *maxV)
{
    OSErr err = noErr;
    PrintSelectionOnlyContextPtr myContext = NULL;

    // this is a Pascal string because it is the string
    // that goes into a menu item
    Str255      theTitle = "\pSample Application Plug-in";

    err = InitContext( &myContext );

    if (err == noErr)
    {
        *context = (PMPDEContext) myContext;

        // Calculate the maximum screen real estate this plug-in needs.
        *maxH = kPrintSelCheckBoxLeftMargin + kPrintSelCheckBoxHSize;
        *maxV = kPrintSelCheckBoxTopMargin + kPrintSelCheckBoxVSize + 6;

        CFStringRef  theTitleRef;
        CFStringRef  theUserOptionKindRef;

        theTitleRef = CFStringCreateWithPascalString(
                        kCFAllocatorSystemDefault,
                        theTitle,
                        CFStringGetSystemEncoding());
```


Preliminary © Apple Computer, Inc. 5/23/00

```
        if (theTitleRef != NULL)
            *title = theTitleRef;

        theUserOptionKindRef = CFStringCreateWithCString(
                                kCFAllocatorSystemDefault,
                                kSampleAppUserOptionKindID,
                                CFStringGetSystemEncoding());
        if (theUserOptionKindRef != NULL)
            *userOptionKind = theUserOptionKindRef;

        // In the next line, use a constant for your own creator type
        *creator = 'spde';

    }
    else        // return an error
        err = kPMErrPDEInvalidContext;

    return (err);
}
```

MyPDEInitialize

`Initialize` — Called by the Mac OS X Printing Manager after it calls the Prologue function. Your initialization routine uses information supplied by the Printing Manager to initialize its user option, and may supply further information about the PDE. (page 63)

The previous paragraph describes a generic `Initialize` function. The version below, implemented by the sample application PDE, creates the controls (a checkbox) for its user option and embeds them in the passed user pane. It also returns flags information indicating it doesn't support any special features. Finally, it calls the PDE's own `MyPDESync` (page 42) function so it can initialize its user option controls from the ticket settings for the print job.

Listing 1-15 A sample PDE Initialize function

```
static OSStatus
MyPDEInitialize( PMPDEContext        context,
                 PMPDEFlags          *flags,
```


Preliminary © Apple Computer, Inc. 5/23/00

```
                PMPDERef            ref,
                ControlRef          embedderUserPane,
                PMPrintSession      printSession)
{
    OSStatus                        err = noErr;
    PrintSelectionOnlyContextPtr    myContext = NULL;
                                    // Pointer to global data block.

    myContext = (PrintSelectionOnlyContextPtr) context;

    if ( (myContext != NULL) && (printSession != NULL) )
    {
        // get the windowref from the user pane
        WindowRef   theWindow = NULL;
        theWindow = GetControlOwner( embedderUserPane );

        // get the user pane's frame rect
        Rect    frameRect;
        GetControlBounds(embedderUserPane, &frameRect);

        // The user pane's rect is the rect we should use to draw our
        // controls into. The printing system calculates the user pane
        // size based on the maxh and maxv sizes returned by our
        // MyPDEPrologue function.
        SetRect(&(myContext->theFrameRect),
                    frameRect.left, frameRect.top,
                    frameRect.right, frameRect.bottom);

        Rect theCheckBoxRect;
        GetPrintSelCheckBoxRect( frameRect, &theCheckBoxRect );
        myContext->thePrintSelTextOnlyControlRef =
                NewControl( theWindow, &theCheckBoxRect,
                    "\pPrint Selected Text Only", false, 0, 0, 1,
                        kControlCheckBoxAutoToggleProc, 0);

        // embed the control in to the user pane; establish control where
        // plug-in's controls are in the window's control hierarchy
        err = EmbedControl(myContext->thePrintSelTextOnlyControlRef,
                            embedderUserPane);

        // Set the control's visibility to visible, so when the user pane is
```


Preliminary © Apple Computer, Inc. 5/23/00

```
        // made visible the plug-in's controls will be shown.
        SetControlVisibility(myContext->thePrintSelTextOnlyControlRef,
                             true, false);

        *flags = kPMPDENoFlags;

        // Initialize this plug-in's controls based on the information in the
        // PageSetup or PrintSettings ticket.
        MyPDESync(context, printSession, true);
    }
    else
        err = kPMErrPDEInvalidContext;

    return (err);
}
```

MyPDESync

`Sync` — Gives the printing dialog extension a chance to synchronize its settings (and control elements) with values in either the PMPageFormat or the PMPrintSettings ticket. Your synchronization function either updates its settings or modifies the ticket settings, depending on the reinitializePlugIn parameter. (page 65)

The previous paragraph describes a generic `Sync` function. The version below, implemented by the sample application PDE, shows how to extract a PDE's user option information from a print ticket.

Listing 1-16 A sample PDE Sync function

```
static OSStatus
MyPDESync(PMPDEContext       context,
        PMPrintSession       printSession,
        Boolean              reinitializePlugin)
{
    OSStatus                          err = noErr;
    PrintSelectionOnlyContextPtr      myContext = NULL;
                                      // Pointer to global data block.

    myContext = (PrintSelectionOnlyContextPtr) context;
```


Preliminary © Apple Computer, Inc. 5/23/00

```
if ( (myContext != NULL) && (printSession != NULL) )
{
    CFNumberRef      cfTicketRef = NULL;
    PMTicketRef      printSettings = NULL;

    // Get the PrintSettings TicketRef from the Session.
    // The TicketRef in the session is wrapped in a CFNumber type.
    err = PMSessionGetDataFromSession(printSession,
             kPMPrintSettingsRef, (CFTypeRef *)&cfTicketRef);
    if ( NULL == cfTicketRef )
        err = kPMErrPDEInvalidPrintSettings;

    // Now lets extract the real PMTicketRef from the CFNumber
    if ( err == noErr )
        err = CFNumberGetValue(cfTicketRef, kCFNumberSInt32Type,
                 (void *)&printSettings);

    if ( ( err == noErr ) && ( printSettings != NULL )  )
    {
        if ( reinitializePlugin )
        {
            Boolean printSelectionOnly;

            // Get the initial value for whether we should print
            // the selected text only.
            err = PMTicketGetBoolean(printSettings, kPMTopLevel,
                     kPMTopLevel,kAppPrintSelectionOnlyKey,
                     &printSelectionOnly);

            if (err == noErr)
            {
                if (printSelectionOnly)
                    SetControlValue(
                        myContext->thePrintSelTextOnlyControlRef,
                        kControlCheckBoxCheckedValue);
                else
                    SetControlValue(
                        myContext->thePrintSelTextOnlyControlRef,
                        kControlCheckBoxUncheckedValue);
            }
```


Preliminary © Apple Computer, Inc. 5/23/00

```
            }
            else
            {
                SInt16  theControlValue = -1;

                theControlValue = GetControlValue(
                            myContext->thePrintSelTextOnlyControlRef);

                if (theControlValue == kControlCheckBoxCheckedValue)
                {
                    // if the tag is not part of the ticket it will be
                    // added and set by the PMTicketSetBoolean routine
                    err = PMTicketSetBoolean(
                                printSettings, kPMPrintingManager,
                                kAppPrintSelectionOnlyKey,
                                (Boolean) true, kPMUnlocked );
                }
                else
                {
                    err = PMTicketSetBoolean(printSettings,
                            kPMPrintingManager, kAppPrintSelectionOnlyKey,
                            (Boolean) false, kPMUnlocked );
                }
            }
        }

        err = kPMErrPDEInvalidPrintSettings;
    }
    else
        err = kPMErrPDEInvalidContext;

    return (err);
}
```

MyPDEGetSummaryText

`GetSummaryText` — Called when the Mac OS X Printing Manager needs to display a summary of the printing dialog extension settings. Your summarization function supplies a title strings and a brief (one line) textual summaries of the current settings in the user option. (page 66)

Preliminary © Apple Computer, Inc. 5/23/00

The previous paragraph describes a generic `GetSummaryText` function. The version below, implemented by the sample application PDE, provides text to describe whether the user option is set to print the selected text only. The Mac OS X Printing Manager displays the summary text in the last panel in the settings panels navigation menu—the pop-up menu that shows the user option "Copies & Pages" in Figure 1-3 (page 23).

Listing 1-17 A sample PDE GetSummaryText function

```
static OSStatus
MyPDEGetSummaryText( PMPDEContext context,
                CFArrayRef *titleArray,
                CFArrayRef *summaryArray )
{
    OSStatus err = noErr;
    PrintSelectionOnlyContextPtr myContext = NULL;
                                // Pointer to global data block.
    myContext = (PrintSelectionOnlyContextPtr) context;

    if (myContext != NULL)
    {
        *titleArray = NULL;
        *summaryArray = NULL;

        CFMutableArrayRef    theTitleArray = NULL;
        CFMutableArrayRef    theSummaryArray = NULL;

        theTitleArray = CFArrayCreateMutable(
                            kCFAllocatorSystemDefault, 0, NULL);
        theSummaryArray = CFArrayCreateMutable(
                            kCFAllocatorSystemDefault, 0, NULL);

        if ( (theTitleArray != NULL) && (theSummaryArray != NULL) )
        {
            // Get the pascal style string from the scale edit text control
            // and put it into the summary text array
            Str255  theSummaryString;
            SInt16  theControlValue = -1;

            theControlValue = GetControlValue(
```


Preliminary © Apple Computer, Inc. 5/23/00

```
                            myContext->thePrintSelTextOnlyControlRef );

            if (theControlValue == kControlCheckBoxCheckedValue)
            {
                BlockMove( "Yes", (Ptr) &theSummaryString[1], 3);
                theSummaryString[0] += 3;
            }
            else
            {
                BlockMove( "Nope, Print All Text",
                                (Ptr) &theSummaryString[1], 20);
                theSummaryString[0] += 20;
            }
            Str255      theTitle = "\pPrint Selected Text Only";
            CFStringRef  title;
            title = CFStringCreateWithPascalString(
                        kCFAllocatorSystemDefault,
                        theTitle,
                        CFStringGetSystemEncoding());
            CFArrayAppendValue(theTitleArray, title);

            CFStringRef  summary;
            summary = CFStringCreateWithPascalString(
                        kCFAllocatorSystemDefault,
                        theSummaryString,
                        CFStringGetSystemEncoding());
            CFArrayAppendValue(theSummaryArray, summary);

            *titleArray = theTitleArray;
            *summaryArray = theSummaryArray;
        }
    }
    else
        err = kPMErrPDEInvalidContext;

    return ( err );
}
```


Preliminary © Apple Computer, Inc. 5/23/00

MyPDEOpen

`Open` — Called at any time after the PDE has been initialized and immediately before the user option is made visible to the user. Your open function performs any required tasks before the user option is opened. (page 67)

The previous paragraph describes a generic `Open` function. The version below, implemented by the sample application PDE, checks for a valid context, but doesn't need to do anything else for its simple user option.

Listing 1-18 A sample PDE Open function

```
static OSStatus MyPDEOpen( PMPDEContextcontext )
{
    OSStatus                        err = noErr;
    PrintSelectionOnlyContextPtr    myContext = NULL;
                                    // Pointer to global data block.

    myContext = (PrintSelectionOnlyContextPtr) context;

    if (myContext != NULL)
    {
        // Do anything special your PDE requires.
    }
    else
        err = kPMErrPDEInvalidContext;

    return ( err );
}
```

MyPDEClose

`Close` — Called at any time after the PDE's Open function has been called, and immediately after the user option is hidden from the user. Your close function performs any required tasks after the user option is closed. (page 68)

The previous paragraph describes a generic `Close` function. The version below, implemented by the sample application PDE, checks for a valid context, but doesn't need to do anything else for its simple user option.

Preliminary © Apple Computer, Inc. 5/23/00

Listing 1-19 A sample PDE Close function

```
static OSStatus MyPDEClose( PMPDEContext context )
{
    OSStatus                        err = noErr;
    PrintSelectionOnlyContextPtr    myContext = NULL;
                                    // Pointer to global data block.

    myContext = (PrintSelectionOnlyContextPtr) context;

    if (myContext != NULL)
    {
        // Do anything special your PDE requires.
    }
    else
        err = kPMErrPDEInvalidContext;

    return ( err );
}
```

MyPDETerminate

`Terminate` — Called by the Mac OS X Printing Manager immediately before closing a printing dialog after a user dismisses the dialog. Your termination function performs any required tasks, such as releasing memory. (page 69)

The previous paragraph describes a generic `Terminate` function. The version below, implemented by the sample application PDE, checks for a valid context, then possibly frees the user option's one control, as well as the context itself.

Listing 1-20 A sample PDE Terminate function

```
static OSStatus MyPDETerminate( PMPDEContext context, OSStatus status  )
{
    OSStatus                        err = noErr;
    PrintSelectionOnlyContextPtr    myContext = NULL;
                                    // Pointer to global data block.

    myContext = (PrintSelectionOnlyContextPtr) context;
```


Preliminary © Apple Computer, Inc. 5/23/00

```
    if (myContext != NULL)
    {

        if ( myContext->thePrintSelTextOnlyControlRef != NULL )
        {
            // Now dispose of the actual control
            DisposeControl( myContext->thePrintSelTextOnlyControlRef );
        }

        // Free the global context.
        if (context != NULL)
        {
            DisposePtr((Ptr) myContext);
            myContext = NULL;
        }
    }
    else
        err = kPMErrPDEInvalidContext;

    return ( err );

}
```

Implementing Additional Functions for the PDE

To implement the user option provided by your PDE, you may need to write additional private functions. The sample application PDE uses the following functions:

- `CreatePlugInInterface` (page 49)
- `InitContext` (page 51)
- `GetPrintSelCheckBoxRect` (page 52)

CreatePlugInInterface

This function creates an instance of the plug-in interface (the function table for the PDE). The interface structure used in this example, `PrintSelOnlyPlugInInterface`, is shown in Listing 1-5 (page 27). It incorporates the `PlugInIntfVTable` (page 73) structure, which in turn incorporates the `PMPlugInHeaderVTable` (page 72) structure.

Preliminary © Apple Computer, Inc. 5/23/00

The result is a structure that includes the header interface functions required by all Mac OS X Printing Manager plug-ins, the callback functions required by all printing dialog extensions, and any additional data specific to your PDE—in other words, a full-fledged PDE interface, including function table and data. This interface mechanism is described in "Printing Dialog Extension Architecture" (page 19).

After instantiating an interface structure and vtable, the `CreatePlugInInterface` function sets all the function references for the function table. It also sets the reference count, the only data field defined by the sample application PDE, to 1.

The `MyCFQueryInterface` (page 34) function calls `CreatePlugInInterface` when it is asked to create an interface for our PDE, as shown in Listing 1-10 (page 34).

Listing 1-21 A sample CreatePlugInInterface function

```
static OSStatus CreatePlugInInterface( PlugInIntf **objPtr )
{
    PrintSelOnlyPlugInInterface*     intf = NULL;
    PlugInIntfVTable*                vtable = NULL;
    OSStatus                         err = noErr;

    // Allocate object and clear it:
    intf = (PrintSelOnlyPlugInInterface*)
                NewPtrClear( sizeof( PrintSelOnlyPlugInInterface ));

    if (intf != NULL)
    {
        // Assign all plug-in data members:
        intf->refCount = 1;

        // Allocate object's vtable and clear it:
        vtable = (PlugInIntfVTable*)
                    NewPtrClear( sizeof( PlugInIntfVTable ));
        if (vtable != NULL)
        {
            intf->vtable = vtable;

            // Assign all plug-in header methods:
            vtable->plugInHeader.Retain          = MyPMRetain;
            vtable->plugInHeader.Release         = MyPMRelease;
```


Preliminary © Apple Computer, Inc. 5/23/00

```
            vtable->plugInHeader.GetAPIVersion    = MyPMGetAPIVersion;

            // Assign all plug-in methods:
            vtable->Prologue         = MyPDEPrologue;
            vtable->Initialize       = MyPDEInitialize;
            vtable->Sync             = MyPDESync;
            vtable->GetSummaryText   = MyPDEGetSummaryText;
            vtable->Open             = MyPDEOpen;
            vtable->Close            = MyPDEClose;
            vtable->Terminate        = MyPDETerminate;

            *objPtr = (PlugInIntf*) intf;
        }
        else
        {
            err = kPMErrPDERefInvalid;
        }
    }
    else
    {
        err = kPMErrPDERefInvalid;
    }
    // Return results:
    return err;
}
```

InitContext

After the Mac OS X Printing Manager obtains a a function table for a printing dialog extension (PDE), it calls the PDE's `MyPDEPrologue` (page 38) function to obtain information about the PDE's user option. The sample application PDE's `MyPDEPrologue` function calls `InitContext` to create and initialize an instance of a context, described in "Defining a Private Context" (page 28). The `InitContext` function merely initializes the context to safe values—the context gets fully initialized when the Printing Manager calls the `MyPDEInitialize` (page 40) function, which in turn calls the `MyPDESync` (page 42) function.

The Printing Manager passes the context supplied by the `MyPDEPrologue` function in subsequent calls to other PDE functions. The context structure defined by the sample PDE is shown in Listing 1-6. This private context stores the user option's drawing area, as well as a reference to the option's lone control, a checkbox.

Preliminary © Apple Computer, Inc. 5/23/00

Listing 1-22 A sample InitContext function

```
static
OSStatus
InitContext( PrintSelectionOnlyContextPtr *context )
{
    OSStatus    err = noErr;         // Error condition.

    /*
    Allocate the global context.
    */
    *context = (PrintSelectionOnlyContextPtr)
         NewPtrClear(sizeof(PrintSelectionOnlyContext));

    if (NULL != *context)
    {
        /*
        Initialize the global data.
        */
        SetRect( &((*context)->theFrameRect), 0,0,0,0);
        (*context)->thePrintSelTextOnlyControlRef = NULL;
    }
    else
        err = MemError();

    return (err);

}   // InitContext
```

GetPrintSelCheckBoxRect

The PDE's `MyPDEInitialize` (page 40) function calls the `GetPrintSelCheckBoxRect` function to obtain the size and location for the sample application PDE's user option.

Listing 1-23 A sample function that obtains user option information

```
static void GetPrintSelCheckBoxRect(Rect inRect, Rect* outRect)
{
```


Preliminary © Apple Computer, Inc. 5/23/00

```
    outRect->left       = inRect.left + kPrintSelCheckBoxLeftMargin;
    outRect->top        = inRect.top + kPrintSelCheckBoxTopMargin;
    outRect->bottom     = outRect->top + kPrintSelCheckBoxVSize;
    outRect->right      = outRect->left + kPrintSelCheckBoxHSize;
}
```

The sample PDE defines user option size and location constants as follows:

```
#define     kPrintSelCheckBoxTopMargin      16
#define     kPrintSelCheckBoxLeftMargin     40
#define     kPrintSelCheckBoxHSize          300
#define     kPrintSelCheckBoxVSize          16
```

Creating a Custom Information Property List

Printing dialog extensions are based on the CFPlugIn model, which is described in the document "Core Foundation Plug-in Services." A CFPlugIn must associate at least one factory ID with the CFPlugInFactories key and at least one type ID with the CFPlugInTypes key. The standard way to create these associations is to store the information in a custom information property list. The property list can then be stored in the plug-in's bundle. You can find information about property lists and bundles in "Core Foundation Property List Services" and "Core Foundation Bundle Services."

The information property list defines various aspects of the plug-in's runtime behavior and can also specify whether the plug-in should be registered dynamically. Listing 1-24 (page 53) shows the custom information property list for the sample application PDE described in this document.

Listing 1-24 A custom information property list for the sample application PDE

```
{
    CFBundleExecutable = "SampleAppPDE";
    CFBundleIdentifier = "SampleAppPDE";
    CFPlugInDynamicRegistration = NO;
    CFPlugInFactories =
    {
       "BED2EC92-E57F-11D3-80C9-0050E4603277" = "SampleAppPDEPluginFactory";
    };
```

```
    CFPlugInTypes =
    {
        "BCB07250-E57F-11D3-8CA6-0050E4603277" =
                                        ("BED2EC92-E57F-11D3-80C9-0050E4603277");
    };
}
```

`CFBundleExecutable`

The sample application PDE supplies the string `"SampleAppPDE"` as the name of its executable.

`CFBundleIdentifier`

The sample application PDE supplies the string `"SampleAppPDE"` as the name of its identifier.

`CFPlugInDynamicRegistration`

PDE plug-ins typically load statically, so the sample application PDE supplies the value `NO` for the dynamic registration key.

`CFPlugInFactories`

This entry in the property list associates a factory ID with the name of the PDE's factory function. The value on the left side of the equal sign is the UUID defined by the `kSampleAppPDEIntfFactoryIDStr` constant, as described in "Defining Factory and Type IDs" (page 23). The string on the right, `"SampleAppPDEPluginFactory"`, is the name of the factory function. This function is shown in Listing 1-7 (page 30).

`CFPlugInTypes`

This entry associates a factory function with a plug-in type. The type ID on the left comes from the constant `kAppPrintDialogTypeIDStr`, which specifies a Print dialog PDE. PDE type constants are described in "Printing Dialog Extension Type ID Constants" (page 77). The factory ID on the right is the same UUID defined by the `kSampleAppPDEIntfFactoryIDStr` constant and used in the previous entry in the property list.

Preliminary © Apple Computer, Inc. 5/23/00

Using a PDE in Your Application

Once you have completed the steps to design and implement a printing dialog extension, it is very easy for your application to use it. Before calling the `PMSessionPrintDialog` function or the `PMSessionPageSetupDialog` function to display a printing dialog, you just call the CFPlugIn function `CFPlugInCreate` and the Mac OS X Printing Manager takes over from there.

For more information on how the Printing Manager works with plug-ins to display user options, see the section "Printing Dialog Extension Architecture" (page 19).

Preliminary © Apple Computer, Inc. 5/23/00

Preliminary © Apple Computer, Inc. 5/23/00

CHAPTER 2

Printing Dialog Extension Reference

This is a preliminary document. While it has received some technical review, there are likely to be changes to some of the information provided here.

The reference material described here is part of the Mac OS X Printing Manager. This material will eventually be incorporated into a larger reference document.

For related material, see the document "Carbon Printing Manager Reference."

Printing Dialog Extension Callback Functions

Printing Manager Plug-In Callback Functions

These callbacks are defined in the `PMPlugInHeaderVTable` (page 72) structure. They must be implemented by all Mac OS X Printing Manager plug-ins, including printing dialog extensions.

Preliminary © Apple Computer, Inc. 5/23/00

- `Retain` — Increments the reference count for an instance of a plug-in. Your retain function increments the count of references to your PDE. (page 58)
- `Release` — Decrements the reference count for an instance of a plug-in. Your release function sets the passed object pointer to NULL, decrements the reference count for your PDE, and if the count reaches 0, frees the PDE and any related storage. (page 59)
- `GetAPIVersion` — Supplies API version information for a plug-in. Your version function supplies information that can be used to determine PDE compatibility. (page 59)

Retain

Increments the reference count for an instance of a plug-in. Your retain function increments the count of references to your PDE.

```
OSStatus (*Retain) ( PMPlugInHeaderInterface *obj );
```

You can use any name for your retain function, but if you were to name it `MyPMRetain` you would declare it like this:

```
OSStatus MyPMRetain ( PMPlugInHeaderInterface *obj);
```

`obj`

A pointer to a header interface of type `PMPlugInHeaderInterface` (page 72). Your retain function can cast this object to your PDE interface type to obtain information specific to your PDE, as shown in Listing 1-11 (page 36).

Discussion

The `Retain` function works with the `Release` (page 59) function to track references to an instance of a plug-in and free the instance when it is no longer referenced.

For an example of a retain function, see Listing 1-11 (page 36).

Preliminary © Apple Computer, Inc. 5/23/00

Release

Decrements the reference count for an instance of a plug-in. Your release function sets the passed object pointer to `NULL`, decrements the reference count for your PDE, and if the count reaches 0, frees the PDE and any related storage.

```
OSStatus (*Release) ( PMPlugInHeaderInterface **objPtr );
```

You can use any name for your release function, but if you were to name it `MyPMRelease` you would declare it like this:

```
OSStatus MyPMRelease(
       PMPlugInHeaderInterface **objPtr);
```

`objPtr`

A pointer to a pointer to a header interface of type `PMPlugInHeaderInterface` (page 72). Your release function can cast this object to your PDE interface type to obtain information specific to your PDE, as shown in Listing 1-12 (page 37).

Discussion

The `Release` function works with the `Retain` (page 58) function to track references to an instance of a plug-in and free the instance when it is no longer referenced.

For an example of a release function, see Listing 1-12 (page 37).

GetAPIVersion

Supplies API version information for a plug-in. Your version function supplies information that can be used to determine PDE compatibility.

```
OSStatus (*GetAPIVersion) ( PMPlugInHeaderInterface *obj,
               PlugInAPIVersion *versionPtr);
```

You can use any name for your get API version function, but if you were to name it `MyPMGetAPIVersion` you would declare it like this:

```
OSStatus MyPMGetAPIVersion(
       PMPlugInHeaderInterface obj,
       PluginAPIVersion *versionPtr);
```


Preliminary © Apple Computer, Inc. 5/23/00

`obj`

A pointer to a header interface of type `PMPlugInHeaderInterface` (page 72). You can cast this interface object to your PDE interface type, as shown in Listing 1-12 (page 37), although you may not need to do so to supply version information.

`versionPtr`

A pointer to a plug-in interface version structure, defined in `PluginAPIVersion` (page 71). Your version function supplies the plug-in's version information in the structure. Version constants are described in "Printing Dialog Extension Interface Version Constants" (page 77).

Discussion

For an example of a get API version function, see Listing 1-13 (page 38).

Printing Dialog Extension Callback Functions

The following functions are defined in the interface structure (function table) for printing dialog extensions, `PlugInIntfVTable` (page 73). All printing dialog extensions must implement these functions.

- `Prologue` — Called by the Mac OS X Printing Manager after it obtains a printing dialog extension (PDE) plug-in interface (or function table). Your prologue function returns information about the PDE's user option and may also provide its own private context information. (page 61)
- `Initialize` — Called by the Mac OS X Printing Manager after it calls the Prologue function. Your initialization routine uses information supplied by the Printing Manager to initialize its user option, and may supply further information about the PDE. (page 63)
- `Sync` — Gives the printing dialog extension a chance to synchronize its settings (and control elements) with values in either the PMPageFormat or the PMPrintSettings ticket. Your synchronization function either updates its settings or modifies the ticket settings, depending on the reinitializePlugIn parameter. (page 65)
- `GetSummaryText` — Called when the Mac OS X Printing Manager needs to display a summary of the printing dialog extension settings. Your summarization function supplies a title strings and a brief (one line) textual summaries of the current settings in the user option. (page 66)

Preliminary © Apple Computer, Inc. 5/23/00

- `Open` — Called at any time after the PDE has been initialized and immediately before the user option is made visible to the user. Your open function performs any required tasks before the user option is opened. (page 67)
- `Close` — Called at any time after the PDE's Open function has been called, and immediately after the user option is hidden from the user. Your close function performs any required tasks after the user option is closed. (page 68)
- `Terminate` — Called by the Mac OS X Printing Manager immediately before closing a printing dialog after a user dismisses the dialog. Your termination function performs any required tasks, such as releasing memory. (page 69)

Prologue

Called by the Mac OS X Printing Manager after it obtains a printing dialog extension (PDE) plug-in interface (or function table). Your prologue function returns information about the PDE's user option and may also provide its own private context information.

```
OSStatus (*Prologue) ( PMPDEContext *context, OSType *creator, CFStringRef
   *userOptionKind, CFStringRef *title, UInt32 *maxH, UInt32 *maxV );
```

You can use any name for your prologue function, but if you were to name it `MyPDEPrologue` you would declare it like this:

```
OSStatus MyPDEPrologue(
       PMPDEContext *context,
       OSType *creator,
       CFStringRef *userOptionKind,
       CFStringRef *title,
       UInt32 *maxH,
       UInt32 *maxV);
```

`context`

A pointer to the printing dialog extension's private context. The Mac OS X Printing Manager passes the context supplied by your `MyPDEPrologue` function in subsequent calls to other PDE functions.

Your PDE may want to use the storage pointed to by the `context` parameter to identify itself, because in Mac OS X it is possible for more than one printing dialog (and hence more than one instance of a PDE's interface) to be instantiated simultaneously. The PDE can store global data it needs to ensure that it is reentrant, as well as pointers to additional functions it may need to use.

Preliminary © Apple Computer, Inc. 5/23/00

For an example of a context structure, see "Defining a Private Context" (page 28).

`creator`

Your PDE supplies the application's creator type in this parameter.

`userOptionKind`

Your PDE supplies a string identifying the kind of user option in this parameter. This string distinguishes a PDE from other PDEs created by the same application. For an example of a custom user option, see "Defining a User Option Kind ID" (page 24). For universal and standard user options, see "Universal and Standard User Option Kind ID Constants" (page 79).

`title`

For a custom user option, your PDE supplies the localized title in this parameter. For universal and standard options, the Mac OS X Printing Manager supplies the title, ignoring any text you supply here.

`maxH`

Your PDE supplies the maximum horizontal extent, in pixels, required to draw its user option. If the Mac OS X Printing Manager cannot provide the area specified by `maxH` and `maxV`, it calls your `Terminate` (page 69) function, passing an error value.

`maxV`

Your PDE supplies the maximum vertical extent, in pixels, required to draw its user option. If the Mac OS X Printing Manager cannot provide the area specified by `maxH` and `maxV`, it calls your `Terminate` (page 69) function, passing an error value.

Discussion

The Mac OS X Printing Manager guarantees that it will call the `Prologue` functions of any registered printing dialog extensions in the order in which their interfaces were supplied by their factory functions. For more information on interfaces and factory functions, see the document "Core Foundation Plug-In Services." For an example of a factory function, see Listing 1-7 (page 30).

For an example of a prologue function, see Listing 1-14 (page 39).

Preliminary © Apple Computer, Inc. 5/23/00

Initialize

Called by the Mac OS X Printing Manager after it calls the `Prologue` function. Your initialization routine uses information supplied by the Printing Manager to initialize its user option, and may supply further information about the PDE.

```
OSStatus (*Initialize) ( PMPDEContext context, PMPDEFlags *flags, PMPDERef
   ref, ControlRef embedderUserPane, PMPrintSession printSession );
```

You can use any name for your initialize function, but if you were to name it `MyPDEInitialize` you would declare it like this

```
OSStatus MyPDEInitialize(
       PMPDEContext context
       PMPDEFlags *flags,
       PMPDERef ref,
       ControlRef embedderUserPane,
       PMPrintSession printSession);
```

`context`

The printing dialog extension's private context (typically a pointer to a context structure defined by you). Your PDE supplies the context when the Mac OS X Printing Manager calls its `Prologue` (page 61) function.

The context may contain anything you choose to store, such as information to identify the PDE, global data needed to ensure the PDE is reentrant, or pointers to additional functions.

For related information, see "Defining a Private Context" (page 28).

`flags`

The `flags` parameter contains 32 bits which indicate to the Mac OS X Printing Manager various properties of the printing dialog extension being registered. Your `MyPDEInitialize` function sets flag bits using the constants defined in "Feature Request Flags Constants" (page 78).

`ref`

A unique reference to the printing dialog extension, supplied by the Mac OS X Printing Manager. You can cast this value to a pointer to your interface structure, such as the structure shown in Listing 1-5 (page 27).

Preliminary © Apple Computer, Inc. 5/23/00

`embedderUserPane`

A reference to a user pane control into which the printing dialog extension is expected to embed its human interface elements. The PDE must embed all user option controls and interface elements into the pane.

`printSession`

A pointer to a printing session. Your PDE can obtain `PMPageFormat` and `PMPrintSettings` ticket information from this parameter.

Discussion

After calling the `Prologue` function, the Mac OS X Printing Manager calls the printing dialog extension's `Initialize` function, providing the PDE with information it can use to initialize the user option, and obtaining further information about the printing dialog extension.

During the execution of `Initialize`, the PDE does anything it needs to do to initialize itself, based upon the information supplied by the Printing Manager. For example, it may wish to adjust its user option interface elements to fit within the available space in the pane before returning the item list of its controls to the Printing Manager.

If any control requires special handling, your user option can install a special Carbon event handler for that control.

Once its interface elements have been embedded into the supplied user pane control, the controls should be made visible, though they do not immediately appear because the user pane is invisible at this time.

Under no circumstances should the PDE attempt to manipulate or make visible the user pane itself!

The PDE should not attempt to draw into the area occupied by the user pane at this point. The user option's controls will be drawn automatically by the Control Manager when the user pane in which they are embedded is made visible.

For an example of an initialization function, see Listing 1-15 (page 40).

Preliminary © Apple Computer, Inc. 5/23/00

Sync

Gives the printing dialog extension a chance to synchronize its settings (and control elements) with values in either the `PMPageFormat` or the `PMPrintSettings` ticket. Your synchronization function either updates its settings or modifies the ticket settings, depending on the `reinitializePlugIn` parameter.

```
OSStatus (*Sync) ( PMPDEContext context, PMPrintSession printSession, Boolean
   reinitializePlugIn );
```

You can use any name for your synchronization function, but if you were to name it `MyPDESync` you would declare it like this:

```
OSStatus MyPDESync(
       PMPDEContext context,
       PMPrintSession printSession,
       Boolean reinitializePlugIn);
```

`context`

The printing dialog extension's private context (typically a pointer to a context structure defined by you). Your PDE supplies the context when the Mac OS X Printing Manager calls its `Prologue` (page 61) function.

The context may contain anything you choose to store, such as information to identify the PDE, global data needed to ensure the PDE is reentrant, or pointers to additional functions.

For related information, see "Defining a Private Context" (page 28).

`printSession`

A pointer to a printing session. Your PDE can obtain `PMPageFormat` and `PMPrintSettings` ticket information from this parameter.

`reinitializePlugIn`

If this Boolean value is `TRUE`, your PDE should obtain `PMPageFormat` or `PMPrintSettings` ticket information from the `printSession` parameter, update its internal settings, and if necessary, update its control elements. If the value is `FALSE`, your PDE should obtain the `PMPageFormat` or `PMPrintSettings` ticket from the `printSession` parameter and set the ticket to match the PDE's internal settings.

Discussion

The Mac OS X Printing Manager typically calls your `MyPDESync` function in the following situations:

Preliminary © Apple Computer, Inc. 5/23/00

- The user has switched out of the user option implemented by your PDE or has closed the printing dialog (not cancelled it).

  In either of these cases, the value of the `reinitializePlugIn` parameter is `FALSE` and your PDE should update the `PMPageFormat` or `PMPrintSettings` ticket, obtained from the `printSession` parameter, to match the PDE's internal settings.

- The user has changed printers and your PDE needs to reinitialize itself from a print ticket associated with the new printer.

  In this case, the value of the `reinitializePlugIn` parameter is `TRUE` and your PDE should update its internal settings to match the `PMPageFormat` or `PMPrintSettings` ticket, obtained from the `printSession` parameter.

You may also choose to call your `MyPDESync` function from the PDE itself. For example, the sample PDE described in this document calls `MyPDESync` from its `MyPDEInitialize` function, as shown in Listing 1-15 (page 40).

For an example of a synchronization function, see Listing 1-16 (page 42).

GetSummaryText

Called when the Mac OS X Printing Manager needs to display a summary of the printing dialog extension settings. Your summarization function supplies a title strings and a brief (one line) textual summaries of the current settings in the user option.

```
OSStatus (*GetSummaryText) ( PMPDEContext context, CFArrayRef *titleArray,
   CFArrayRef *summaryArray );
```

You can use any name for your summary function, but if you were to name it `MyPDEGetSummaryText` you would declare it like this:

```
OSStatus MyPDEGetSummaryText(
        PMPDEContext context,
        CFArrayRef *titleArray,
        CFArrayRef *summaryArray);
```

`context`

The printing dialog extension's private context (typically a pointer to a context structure defined by you). Your PDE supplies the context when the Mac OS X Printing Manager calls its `Prologue` (page 61) function.

Preliminary © Apple Computer, Inc. 5/23/00

The context may contain anything you choose to store, such as information to identify the PDE, global data needed to ensure the PDE is reentrant, or pointers to additional functions.

For related information, see "Defining a Private Context" (page 28).

`titleArray`

A pointer to an array reference. Your summary function stores in this array the title for each setting in the user option. You allocate the title strings you store in the array.

`summaryArray`

A pointer to an array reference. Your summary function stores in this array a brief (one line) textual summary for each setting in the user option. You allocate the title strings you store in the array.

Discussion

When the user chooses "Summary" in the settings panels navigation menu—the same pop-up menu that shows the user option "Copies & Pages" in Figure 1-3 (page 23)—the Mac OS X Printing Manager calls the printing dialog extension's `GetSummaryText` function. The summary text your PDE provides is displayed in the "Summary" panel, the last panel specified in the panels navigation menu.

For an example of a summarization function, see Listing 1-17 (page 45).

Open

Called at any time after the PDE has been initialized and immediately before the user option is made visible to the user. Your open function performs any required tasks before the user option is opened.

```
OSStatus (*Open) ( PMPDEContext context );
```

You can use any name for your open function, but if you were to name it `MyPDEOpen` you would declare it like this:

```
OSStatus MyPDEOpen(
       PMPDEContext context);
```

`context`

The printing dialog extension's private context (typically a pointer to a context structure defined by you). Your PDE supplies the context when the Mac OS X Printing Manager calls its `Prologue` (page 61) function.

Preliminary © Apple Computer, Inc. 5/23/00

The context may contain anything you choose to store, such as information to identify the PDE, global data needed to ensure the PDE is reentrant, or pointers to additional functions.

For related information, see "Defining a Private Context" (page 28).

Discussion

There is no required functionality for the `Open` function.

A user option can either be open or closed. The user interface elements of an open user option are visible to the user.

For an example of an open function, see Listing 1-18 (page 47).

Close

Called at any time after the PDE's `Open` function has been called, and immediately after the user option is hidden from the user. Your close function performs any required tasks after the user option is closed.

```
OSStatus (*Close) ( PMPDEContext context );
```

You can use any name for your close function, but if you were to name it `MyPDEClose` you would declare it like this:

```
OSStatus MyPDEClose(
        PMPDEContext context);
```

`context`

The printing dialog extension's private context (typically a pointer to a context structure defined by you). Your PDE supplies the context when the Mac OS X Printing Manager calls its `Prologue` (page 61) function.

The context may contain anything you choose to store, such as information to identify the PDE, global data needed to ensure the PDE is reentrant, or pointers to additional functions.

For related information, see "Defining a Private Context" (page 28).

Discussion

There is no required functionality for the `Close` function.

Preliminary © Apple Computer, Inc. 5/23/00

The `Close` function allows the PDE to perform any required task before the user option is closed. If, for example, the PDE displays some form of animation using a thread when the user option is open, it might want to stop executing the thread that updates the animation when the user option is not visible.

A user option can either be open or closed. The user interface elements of a closed user option are not visible (except for those managed by the Mac OS X Printing Manager, such as the title).

For an example of a close function, see Listing 1-19 (page 48).

Terminate

Called by the Mac OS X Printing Manager immediately before closing a printing dialog after a user dismisses the dialog. Your termination function performs any required tasks, such as releasing memory.

```
OSStatus (*Terminate) ( PMPDEContext context, OSStatus status );
```

You can use any name for your termination function, but if you were to name it `MyPDETerminate` you would declare it like this:

```
OSStatus MyPDETerminate(
       PMPDEContext context,
       OSStatus status);
```

`context`

The printing dialog extension's private context (typically a pointer to a context structure defined by you). Your PDE supplies the context when the Mac OS X Printing Manager calls its `Prologue` (page 61) function.

The context may contain anything you choose to store, such as information to identify the PDE, global data needed to ensure the PDE is reentrant, or pointers to additional functions.

For related information, see "Defining a Private Context" (page 28).

`status`

Indicates the conditions under which the printing dialog extension was terminated. If, for example, an out-of-memory error occurs during initialization of the PDE, the Mac OS X Printing Manager

Preliminary © Apple Computer, Inc. 5/23/00

might call the `Terminate` function and pass `memFullErr` in the `status` parameter. The PDE should not respond with an alert—the Printing Manager will already have supplied one if necessary.

Error codes specific to printing dialog extensions are shown in "Printing Dialog Extension Result Codes" (page 79).

Discussion

When the user dismisses a printing dialog, the Mac OS X Printing Manager calls each printing dialog extension's termination function immediately before closing the dialog. The PDEs then perform any necessary termination tasks. The dialog can be assumed to be invisible to the user, but still present. The PDE's controls have not yet been deleted.

For an example of a termination function, see Listing 1-20 (page 48).

Printing Dialog Extension Data Types

- `PluginAPIVersion` — Stores the version information for a plug-in's interface. (page 71)
- `PMPlugInHeaderInterface` — Provides access to a vtable that must be the first element in the interface definition for every Mac OS X Printing Manager plug-in. (page 72)
- `PMPlugInHeaderVTable` — Defines a vtable whose functions must be implemented by every Mac OS X Printing Manager plug-in. (page 72)
- `PluginIntf` — Provides access to the full interface that all printing dialog extensions must implement. (page 73)
- `PlugInIntfVTable` — Combines plug-in and PDE vtables to define the full interface that all printing dialog extensions must implement. (page 73)
- `PMPDERef` — Specifies a reference to an instance of a printing dialog extension. (page 75)
- `PMPDEContext` — Specifies a reference to a printing dialog extension's private context. (page 75)

Preliminary © Apple Computer, Inc. 5/23/00

- PMPDEFlags — Specify properties of a printing dialog extension to the Mac OS X Printing Manager. (page 75)

PluginAPIVersion

Stores the version information for a plug-in's interface.

```
typedef struct PluginAPIVersion
   {
       UInt32 buildVersionMajor;
       UInt32 buildVersionMinor;

       UInt32 baseVersionMajor;
       UInt32 baseVersionMinor;
   }
   PluginAPIVersion;
```

Field descriptions

buildVersionMajor

The major component of the API version the plug-in was compiled with.

buildVersionMinor

The minor component of the API version the plug-in was compiled with.

baseVersionMajor

The major component of the base API version this plug-in is upwardly compatible with. That is, the plug-in is guaranteed to be a superset of all versions of the API starting with this one.

baseVersionMinor

The major component of the base API version this plug-in is upwardly compatible with. That is, the plug-in is guaranteed to be a superset of all versions of the API starting with this one.

Discussion

In the plug-in model defined by Core Foundation Plug-in Services, an interface cannot ever change once it has been published. Instead of changing a type's interface, you simply add a new interface (more functions) to the type with the changed or additional functionality.

This API version structure shown here provides a mechanism for keeping track of plug-in versions. For related information, see GetAPIVersion (page 59) and "Printing Dialog Extension Interface Version Constants" (page 77).

Preliminary © Apple Computer, Inc. 5/23/00

PMPlugInHeaderInterface

Provides access to a vtable that must be the first element in the interface definition for every Mac OS X Printing Manager plug-in.

```
struct PMPlugInHeaderInterface
   {
       const PMPlugInHeaderVTable *vtable;
   };
```

Field descriptions

`vtable`

A pointer to a structure of type `PMPlugInHeaderVTable` (page 72). This structure defines an interface whose functions every Mac OS X plug-in must implement. This structure must also be the first element in any interface definition (except the `IUnknown` interface defined in CFPlugin.h).

Discussion

Some plug-in functions, such as `Retain` (page 58) and `Release` (page 59), pass a pointer to a `PMPlugInHeaderInterface` structure, which your application can cast to a pointer to its own PDE interface structure, as shown in Listing 1-11 (page 36).

PMPlugInHeaderVTable

Defines a vtable whose functions must be implemented by every Mac OS X Printing Manager plug-in.

```
struct PMPlugInHeaderVTable
   {
       OSStatus Retain ( PlugInInterface* obj );
       OSStatus Release ( PlugInInterface** objPtr );
       OSStatus GetAPIVersion ( PlugInInterface* obj,
                           PluginAPIVersion* versionPtr );
   };
```

Field descriptions

`Retain`

For a description of this function, see `Retain` (page 58).

`Release`

For a description of this function, see `Release` (page 59).

`GetAPIVersion`

For a description of this function, see `GetAPIVersion` (page 59).

Preliminary © Apple Computer, Inc. 5/23/00

Discussion

All Mac OS X Printing Manager plug-ins, including printing dialog extensions, must implement the interface defined in the `PMPlugInHeaderVTable` structure. PDEs must also implement additional interface elements, defined in the `PlugInIntfVTable` (page 73) structure.

PlugInIntf

Provides access to the full interface that all printing dialog extensions must implement.

```
struct PlugInIntf
   {
       PlugInIntfVTable* vtable;
   };
```

Field descriptions

`vtable`

A pointer to a vtable, or table of functions, of type `PlugInIntfVTable` (page 73). The first field in all PDE interface objects must be of type `PMPlugInHeaderVTable` (page 72). That is the case here, because `vtable` is a pointer to a structure of type `PlugInIntfVTable`, whose first field is a structure of type `PMPlugInHeaderVTable`.

Discussion

A PDE is an instance of this structure. The IUnknown interface for the sample PDE creates such an instance when its `MyCFQueryInterface` function, shown in Listing 1-10 (page 34), calls the `CreatePlugInInterface` function, shown in Listing 1-21 (page 50).

PlugInIntfVTable

Combines plug-in and PDE vtables to define the full interface that all printing dialog extensions must implement.

```
struct PlugInIntfVTable
   {
       PMPlugInHeaderVTable plugInHeader;

       OSStatus (*Prologue) (
               PMPDEContext         *context,
               OSType               *creator,
               CFStringRef          *userOptionKind,
```


Preliminary © Apple Computer, Inc. 5/23/00

```
                CFStringRef             *title,
                UInt32                  *maxH,
                UInt32                  *maxV );
        OSStatus (*Initialize) (
                PMPDEContext            context,
                PMPDEFlags              *flags,
                PMPDERef                ref,
                ControlRef              embedderUserPane,
                PMPrintSession          printSession );
        OSStatus (*Sync) (
                PMPDEContext            context,
                PMPrintSession          printSession,
                Boolean                 reinitializePlugIn);
        OSStatus (*GetSummaryText) (
                PMPDEContext            context,
                CFArrayRef              *titleArray,
                CFArrayRef              *summaryArray );
        OSStatus (*Open(
                PMPDEContext            context );
        OSStatus (*Close) (
                PMPDEContext            context );
        OSStatus (*Terminate) (
                PMPDEContext            context,
                OSStatus                status);
    };
```

Field descriptions

`plugInHeader`

A plug-in header defined in `PMPlugInHeaderVTable` (page 72). The plug-in header is required with all printing dialog extension plug-ins and must be the first field in the interface structure.

`Prologue`

For a description of this function, see `Prologue` (page 61).

`Initialize`

For a description of this function, see `Initialize` (page 63).

`Sync`

For a description of this function, see `Sync` (page 65).

`GetSummaryText`

For a description of this function, see `GetSummaryText` (page 66).

`Open`

For a description of this function, see `Open` (page 67).

`Close`

For a description of this function, see `Close` (page 68).

Preliminary © Apple Computer, Inc. 5/23/00

`Terminate`

For a description of this function, see `Terminate` (page 69).

Discussion

Your printing dialog extension must implement all of the functions defined in the `PlugInIntfVTable` structure. For a complete listing of the functions your PDE must implement, see "Writing Code to Implement Your PDE" (page 29).

For related information, see `PlugInIntf` (page 73).

PMPDERef

Specifies a reference to an instance of a printing dialog extension.

```
typedef SInt32 PMPDERef;
```

Discussion

Many of the functions defined in the `PlugInIntfVTable` (page 73) structure use parameters of this data type to provide access to a PDE.

PMPDEContext

Specifies a reference to a printing dialog extension's private context.

```
typedef UInt32 PMPDEContext;
```

Discussion

The Mac OS X Printing Manager passes the context supplied by your `Prologue` (page 61) function in subsequent calls to other PDE functions.

PMPDEFlags

Specify properties of a printing dialog extension to the Mac OS X Printing Manager.

```
typedef UInt32 PMPDEFlags;
```

Discussion

The `Initialize` (page 63) function supplies the Mac OS X Printing Manager with flag values in a parameter of this type. Possible flag values are defined in "Feature Request Flags Constants" (page 78).

Preliminary © Apple Computer, Inc. 5/23/00

Printing Dialog Extension Constants

- `Printing Dialog Extension Interface ID Constant` — Identifies a plug-in interface as a printing dialog extension. (page 76)
- `Printing Dialog Extension Type ID Constants` — Identify the plug-in type of a printing dialog extension. (page 77)
- `Printing Dialog Extension Interface Version Constants` — Specify the version of a printer dialog extension plug-in's interface. (page 77)
- `Feature Request Flags` — Specify properties of a printing dialog extension to the Mac OS X Printing Manager. (page 78)

Printing Dialog Extension Interface ID Constants

Identifies a plug-in interface as a printing dialog extension.

```
#define kDialogExtensionIntfIDStr "A996FD7E-B738-11D3-8519-0050E4603277"
```

Constant descriptions

`kDialogExtensionIntfIDStr`

Identifies a plug-in interface as being for a printing dialog extension. This constant is defined with a "#define" statement to prevent having the value mangled by a C++ compiler.

Discussion

An interface defines an area of functionality to be implemented by the plug-in developer. For more information on plug-in interfaces, see the document "Core Foundation Plug-In Services."

For information on using an interface type in your PDE, see "Creating a Custom Information Property List" (page 53). For sample code that uses this constant, see Listing 1-10 (page 34).

Preliminary © Apple Computer, Inc. 5/23/00

Printing Dialog Extension Type ID Constants

Identify the plug-in type of a printing dialog extension.

```
#define kAppPageSetupDialogTypeIDStr      "B9A0DA98-E57F-11D3-9E83-0050E4603277"
#define kAppPrintDialogTypeIDStr          "BCB07250-E57F-11D3-8CA6-0050E4603277"
#define kPrinterModuleTypeIDStr           "BDB091F4-E57F-11D3-B5CC-0050E4603277"
```

Constant descriptions

`kAppPageSetupDialogTypeIDStr`

Specifies a PDE plug-in type as a Page Setup dialog PDE supplied by an application.

`kAppPrintDialogTypeIDStr`

Specifies a PDE plug-in type as a Print dialog PDE supplied by an application. For use of this constant in a PDE, see Listing 1-7 (page 30) and "Creating a Custom Information Property List" (page 53).

`kPrinterModuleTypeIDStr`

Specifies a PDE plug-in type as a PDE supplied by a printer module. This type is not used by application PDE developers. Printer modules can't extend a Page Setup dialog, so this constant indicates the PDE is for a Print dialog.

Discussion

A plug-in type is an aggregation of one or more interfaces, each of which defines an area of functionality to be implemented by the plug-in developer. For more information, see the document "Core Foundation Plug-In Services."

These constants are defined with "#define" statements to prevent having the values mangled by a C++ compiler.

Printing Dialog Extension Interface Version Constants

Specify the version of a printer dialog extension plug-in's interface.

```
#define kPDEBuildVersionMajor    1
#define kPDEBuildVersionMinor    0
#define kPDEBaseVersionMajor     1
#define kPDEBaseVersionMinor     0
```


Preliminary © Apple Computer, Inc. 5/23/00

Constant descriptions

`kPDEBuildVersionMajor`

Update this value when you append new APIs to the end of the interface. For example, going from 1 to 2 represents a major change (such as 1.0 to 2.0).

`kPDEBuildVersionMinor`

Update this value when you append new APIs to the end of the interface. For example, going from 0 to 1 represents a minor change (such as 1.0 to 1.1).

`kPDEBaseVersionMajor`

Update this value when you cause some APIs in the interface to become obsolete. Making APIs obsolete breaks the upward compatibility chain for plug-ins and is strongly discouraged.

`kPDEBaseVersionMinor`

Update this value when you cause some APIs in the interface to become obsolete. Making APIs obsolete breaks the upward compatibility chain for plug-ins and is strongly discouraged.

Discussion

For related information, see `GetAPIVersion` (page 59) and `PluginAPIVersion` (page 71).

Feature Request Flags Constants

Specify properties of a printing dialog extension to the Mac OS X Printing Manager.

```
enum
   {
       kPMPDENoFlags    = 0x00000000,
       kPMPDENoSummary  = 0x00000001,
       kPMPDEAllFlags   = (UInt32) -1
   };
```

Constant descriptions

`kPMPDENoFlags`

All flag bits are off, indicating no special features are supported. Until other flag bits are defined, you should use this constant unless your PDE doesn't support summary text. For sample code that uses this constant, see Listing 1-15 (page 40).

Preliminary © Apple Computer, Inc. 5/23/00

`kPMPDENoSummary`

The printing dialog extension won't provide a summary string. If a PDE sets the bit indicated by the `kPMPDENoSummary` flag in response to a call to its `Initialize` (page 63) function, the Mac OS X Printing Manager never calls the PDE's `GetSummaryText` (page 66) function. The PDE should set this flag only if summary information is unnecessary or doesn't apply.

`kPMPDEAllFlags`

All flag bits are on. Until other flag bits are defined, you should not use this constant.

Universal and Standard User Option Kind ID Constants

Identify printing dialog extensions (PDEs) that implement universal and standard user options.

```
//To be supplied.
```

Discussion

User option kind definitions for universal and standard options have not yet been released.

You can create PDEs that override the universal user options provided by the Mac OS X Printing Manager, such as the Copies & Pages user option. You can also create PDEs that implement the standard user options defined but not implemented by the Printing Manager, such as "Color" or "Paper Source." User options are described in "Printing Dialog Extensions and User Options" (page 13).

Printing Dialog Extension Result Codes

Table 2-1 Printing dialog extension (PDE) result codes defined by the Mac OS X Printing Manager

Name	Value	Description
`kPMErrPDEInvalidContext`	-9520	Invalid context for a PDE

Preliminary © Apple Computer, Inc. 5/23/00

Table 2-1 Printing dialog extension (PDE) result codes defined by the Mac OS X Printing Manager

Name	Value	Description
kPMErrPDEInvalidSession	-9521	Invalid session
kPMErrPDEInvalidPrintSettings	-9522	Invalid print settings
kPMErrPDEInvalidPageFormat	-9523	Invalid page format
kPMErrPDEInvalidJobTemplate	-9524	Invalid job template
kPMErrPDEInvalidPrinterInfo	-9525	Invalid printer information
kPMErrPDERefInvalid	-9526	Invalid reference to a PDE
kPMErrPDEGeneralError	-9528	Unspecified PDE error

Preliminary © Apple Computer, Inc. 5/23/00