



Acrobat Interapplication Communication Overview

Adobe Developer Relations

Technical Note #5164



Revised: March 9, 1999

Copyright 1995-1999 Adobe Systems Incorporated. All rights reserved.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated.

No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher.

PostScript is a trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Except as otherwise stated, any reference to a "PostScript printing device," "PostScript display device," or similar item refers to a printing device, display device or item (respectively) that contains PostScript technology created or licensed by Adobe Systems Incorporated and not to devices or items that purport to be merely compatible with the PostScript language.

Adobe, the Adobe Logo, Acrobat, the Acrobat Logo, Acrobat Capture, Acrobat Exchange, Distiller, PostScript, and the PostScript logo are trademarks of Adobe Systems Incorporated.

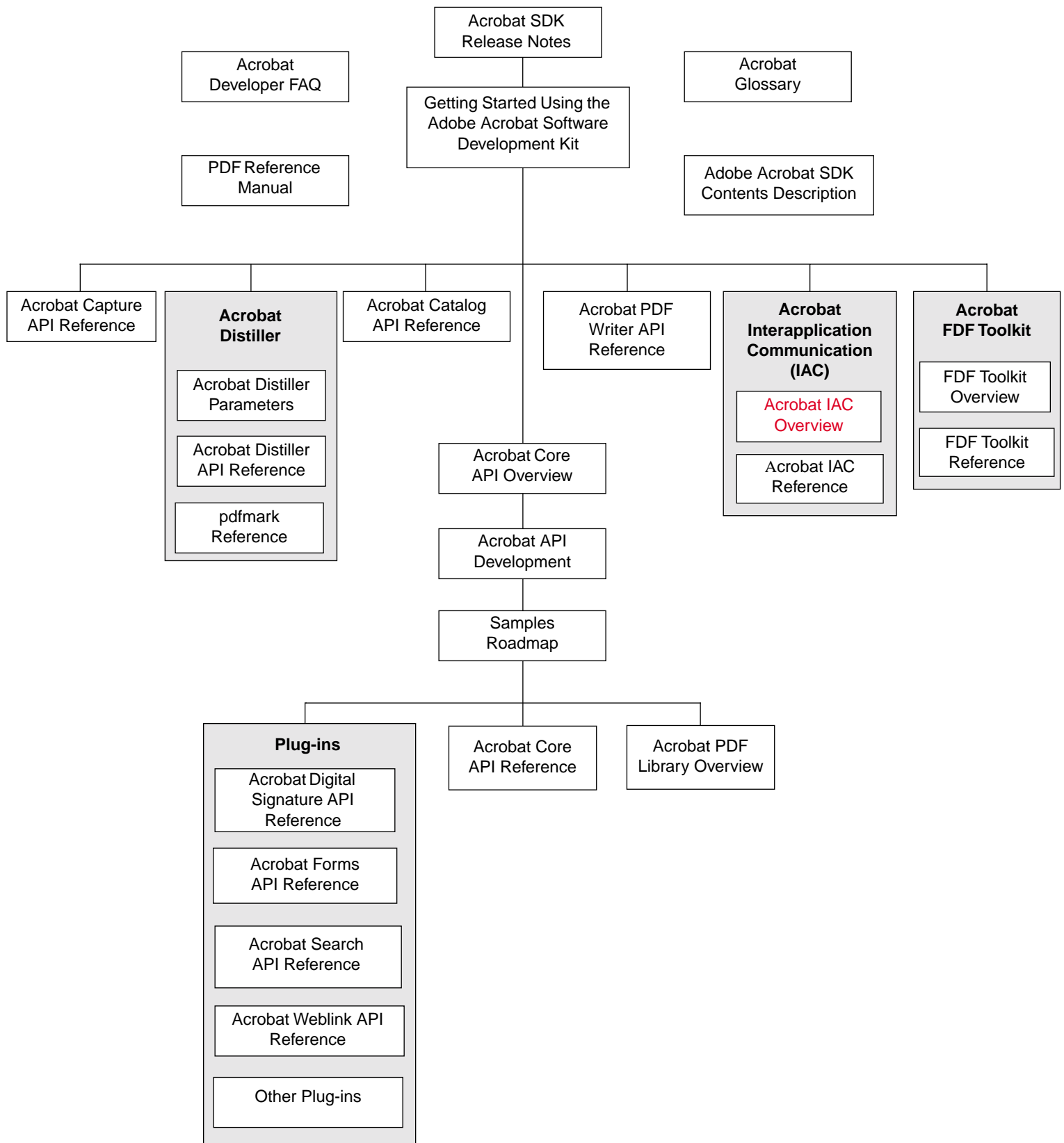
Apple, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. HP-UX is a registered trademark of Hewlett-Packard Company. AIX and PowerPC are registered trademarks of IBM Corporation in the United States. ActiveX, Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries. UNIX is a registered trademark of The Open Group. All other trademarks are the property of their respective owners.

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and non-infringement of third party rights.

Version History		
7 December 1999	Tim Bienz	First 2.0 version.
10 April 1995	Tim Bienz	Minor updates.
15 September 1995	Gary Staas	First 2.1 version; IAC Technical notes split into Overview and On-line Reference documents.
31 January 1996	Gary Staas	Added OLE material.
13 February 1996	Gary Staas	Added UNIX 2.1 information.
9 April 1996	Gary Staas	Minor updates.
24 May 1996	Gary Staas	3.0 beta version.
18 July 1996	Gary Staas	3.0 beta version.

Version History		
13 November 1996	Gary Staas	First 3.0 version.
14 February 1997	Gary Staas	Minor corrections.
3 June 1997	Gary Staas	3.0J and 3.01 version.
12 August 1997	Gary Staas	Added information on new DDE methods.
3 October 1997	Gary Staas	Clarified how to get an LPDISPATCH for objects.
13 February 1998	Gary Staas	Added PDDoc.SetOpenInfo.
22 January 1999	David Downey	Updated 4.0 version.

Documentation Roadmap





Contents

Chapter 1: Introduction 7

- 1.1 IAC Overview 8
- 1.2 Using IAC with Plug-ins 9
- 1.3 Organization of This Document 10
- 1.4 Other Useful Documents 10
- 1.5 Code Samples 11

Chapter 2: Overview 13

- 2.1 IAC Objects 13
 - Acrobat Viewer Level 14
 - PDF File Level 16
- 2.2 Using Objects 18
 - Acrobat Viewer and PDF File Levels 18
 - Bridge Methods 18
 - Limitations 19

Chapter 3: Apple Event Support 20

- 3.1 Differences Among the Acrobat Viewers 21
- 3.2 Objects and Properties 21
- 3.3 Acrobat Viewer Apple event Objects 22
- 3.4 Apple event Suite Support 23
 - Required Suite 23
 - Core Suite 24
 - Acrobat Viewer Suite 24

Chapter 4: OLE Support 27

- 4.1 Differences Among the Acrobat Viewers 27
- 4.2 OLE Server Support 28
- 4.3 OLE Automation Support 28
 - What's Possible: Examples and Approaches 30
 - Development Environments 31
 - Using Visual Basic with the Acrobat SDK 32
 - Understanding the MFC Interfaces to OLE 35
- 4.4 Development Information 43
 - Using OLE Messages 43
 - MDI Usage 44
 - Event Handling 44
 - Using OLE Automation With Visual Basic 45
- 4.5 OLE Automation Summary 45
 - Objects 46
 - Data Types 46
 - Methods 47

Chapter 5: DDE Support 56

- 5.1 Differences Among the Acrobat Viewers 56
- 5.2 General Information 56
- 5.3 Acrobat Viewer DDE Messages 57
 - Application Configuration 58
 - Document Manipulation 58
 - Document Printing 59
 - View Manipulation 60
 - Search-related 60

Appendix A: Coordinate Systems 61

Appendix B: Changes Since Earlier Versions 63

Introduction

This document describes the Interapplication Communication (IAC) support provided by version 4.0 of the Adobe® Acrobat® viewers (Acrobat and Reader). The viewers support a range of IAC through Apple events and AppleScript (on the Apple Macintosh computer), and through DDE and OLE 2 (under Microsoft® Windows®). The capabilities provided are a subset of those provided through the Acrobat core API. The IAC support methods and events essentially serve as “wrappers” for some of the API calls available in the full plug-in API. For a description of the plug-in API, see Technical Note #5190, [Acrobat Core API Overview](#), and Technical Note #5191, [Acrobat Core API Reference](#), which are included with the Acrobat Plug-in SDK.

There is no IAC support in the UNIX® release of Acrobat. Plug-in developers could use native system capabilities (pipes, XtAddInput, and so on) and a plug-in to provide the equivalent communication.

Just as for the plug-in API, IAC support differs between Acrobat and the Reader.

Readers of this document are expected to already understand at least one of the underlying technologies: Apple events, AppleScript, DDE, or OLE. See “[Other Useful Documents](#)” for a list of documents that describe these technologies if you are not familiar with them. In addition, some familiarity with the Acrobat core API, as described in Technical Notes #5190 and #5191, is useful both for understanding the objects used in the IAC interface and because many of the IAC messages are similar to methods present in the plug-in API.

1.1 IAC Overview

The Acrobat viewers' IAC support allows another program to control an Acrobat viewer in the same ways a user could. In addition, the Acrobat viewers can be commanded to render a PDF file into any specified window instead of the Acrobat window. The OLE support provided by the Windows versions of the Acrobat viewers includes both OLE server and OLE automation.

Both Apple events and OLE automation communicate with the viewer through a subset of the objects available in the viewer. Apple events and OLE support slightly different subsets of these objects

The following are objects in the viewer:

- Application — the Acrobat viewer application itself.
- Document — a single open document.
- AVPageView — the view of the document in its window. (Documents that aren't visible don't have AVPageViews.)
- AVMenu — a menu (Apple events only).
- AVMenuItem — a single item in a menu (Apple events only).

The following are objects in PDF documents:

- PDPage — a single page.
- PDDoc — the underlying PDF representation of a document (OLE automation only).
- PDAnnot — an annotation on a page of a document. The PDTextAnnot and PDLinkAnnot classes are two specific annotation types.
- PDTextAnnot — a text annotation. Can only be used as the target of a make event. All other access is via the PDAnnot class (Apple events only).
- PDLinkAnnot — a link annotation. Can only be used as the target of a make event. All other access is via the PDAnnot class (Apple events only).

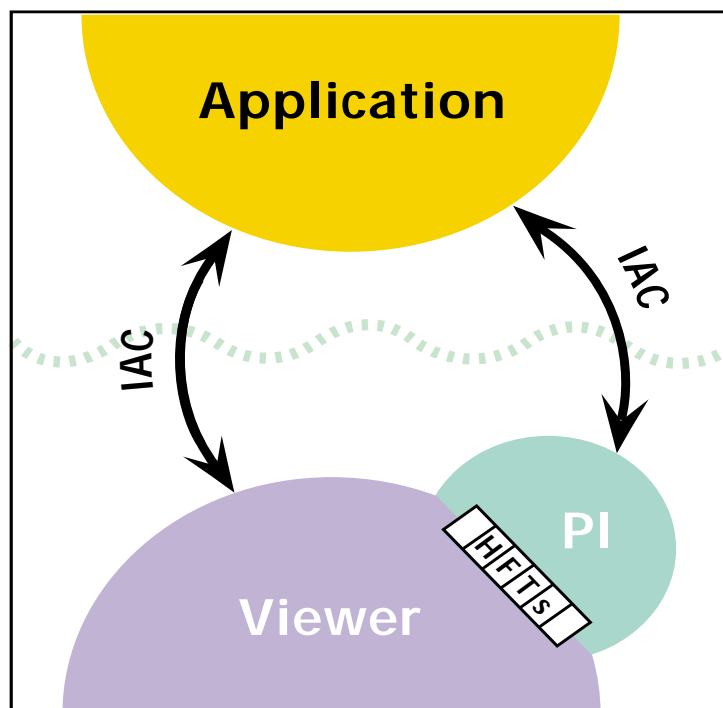
- PDBookmark — a bookmark.
- PDTextSelect — a selection of text, as if a user had used the mouse to select a region of text (OLE automation only).

The names and access methods for these objects are described in the chapters on Apple events and OLE automation.

1.2 Using IAC with Plug-ins

Suppose an IAC interface does not provide functionality your application requires—but these functions are available through the Acrobat core API. You can write a plug-in that implements the functions your application needs and provide an IAC interface to access these functions from your application. For example, the Search plug-in has DDE and Apple event interfaces. [Figure 1, “IAC and Plug-ins”](#) illustrates this connection.

Figure 1 IAC and Plug-ins



1.3 Organization of This Document

This document is split into chapters along technology lines: Apple events (Chapter 3), OLE (Chapter 4), and DDE (Chapter 5). Readers need only read the chapter relevant to their own platform and technology. Detailed information on these IAC interfaces, such as event, method, and message descriptions and their associated parameters, are in Technical Note #5165, [Acrobat Interapplication Communication Reference](#).

1.4 Other Useful Documents

Readers are assumed to be familiar with the Acrobat API. The following technical notes provide this information.

[Acrobat Interapplication Communication Reference](#), Technical Note #5165. Provides detailed information on the Apple event, DDE, and OLE support provided by the Acrobat viewers.

[Acrobat Core API Overview](#), Technical Note #5190. Gives an overview of the objects and methods provided by the Acrobat Core API.

[Acrobat Core API Reference](#), Technical Note #5191. Describes in detail the objects and methods provided by the Acrobat plug-in API.

[Portable Document Format Reference Manual](#), Version 1.3, provides a description of the PDF file format, as well as suggestions for producing efficient PDF files. It is intended for application developers who wish to produce PDF files directly.

Inside Macintosh: Interapplication Communication, ISBN 0-201-62200-9, Addison-Wesley. Contains further information on Apple events and some information on scripting.

AppleScript Language Guide, ISBN 0-201-40735-3, Addison-Wesley. Contains further information on the AppleScript language.

Apple Event Registry: Standard Suites, by Apple Developer Technical Publications, Part number 030-1958-A. Contains further information on the core and required Apple events.

OLE 2 Programmer's Reference Volumes One and Two, ISBN 1-55615-628-6 and ISBN 1-55615-629-4, Microsoft Press. Volume One contains much information on OLE 2.0; Volume Two covers OLE Automation.

1.5 Code Samples

The AutoClik MFC Sample and tutorial in Microsoft Visual C++ 1.5 and 4.2. Contains an example of using OLE automation.

In addition, the Acrobat SDK contains several code samples that show how to use various IAC technologies with Acrobat.

Apple events:

- The AEView sample demonstrates controlling various components of Acrobat for the Macintosh.
- The PrintPages Apple script prompts the user for a PDF file name and prints the file.
- The Rotate AppleScript example shows how to drive the Acrobat viewer using Apple events.
- The SelectText AppleScript example demonstrates how to launch the viewer, query the user to open a PDF file, and selection of text within the PDF document.
- The Set AppleScript example demonstrates the launching of the viewer, querying the user to open two different PDF files, and then setting various parameters including turning the splash screen on and off, setting the active document, modifying the zoom factor and type, and changing the page number that is being displayed.

OLE automation:

- Draw and VBDraw show how to render the PDF page contents into another application's window using OLE 2.0. VBDraw does this via Visual Basic, Draw with Visual C++.
- OIW and VBOIW illustrate how to display the AVDoc window of a PDF file within another application's window using OLE 2.0. VBOIW does this via Visual Basic, OIW with Visual C++.

DDE:

- DDEOPEN demonstrates a basic DDE connection to the Acrobat viewer.

See [Samples](#) for more information about these samples.

CHAPTER 2

Overview

Acrobat exposes much of its internal architecture in an object-oriented spirit, although it's not truly object-oriented in implementation. The plug-in interface exposes nearly 50 objects. The IAC interface for Apple events and OLE automation exposes only about 10 objects. These objects map closely to what the user sees in Acrobat.

Note: DDE IAC capabilities are not organized around objects but use DDE messages to the Acrobat viewer.

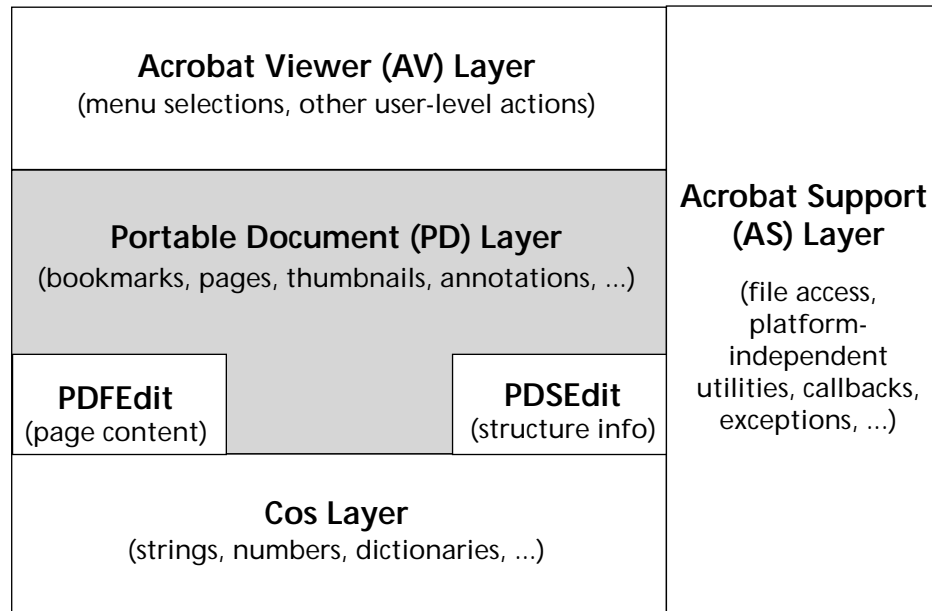
This chapter is a high level survey of the objects Acrobat exposes to developers. See the Apple event, OLE, or DDE chapters for specifics of IAC with Acrobat on your platform.

2.1 IAC Objects

Objects are defined on two levels: objects in the Acrobat viewer such as a view of a document, and objects in the PDF file itself such as a page in a document. See [Figure 2, "Acrobat Viewer Object Levels"](#) for an overview.

Objects, though similar, have different names in Apple events and OLE automation 2.0. These names appear after each object name. The first name is the Apple event name, which you use to create objects through *CreateObjSpecifier* in Apple events or "set ... to" in AppleScript. The second name is the OLE automation 2.0 name used to create an object of that type through *CreateObject* in VisualBasic or *CreateDispatch* in MFC. A name of "None" means there is no corresponding class.

Figure 2 Acrobat Viewer Object Levels



2.1.1 Acrobat Viewer Level

2.1.1.1 Application

Apple event name: *Application*

OLE automation name: *AcroExch.App*

At the top level is the *Application* object. From the application layer, you can control the appearance of Acrobat, whether an Acrobat window appears, and the size of the application window. Your application has access to the menu bar and the toolbar through this object. The application layer also provides access to the visual representation of a PDF file on the screen, that is, an *AVDoc*.

2.1.1.2 AVDoc

Apple event name: *Document*

OLE automation name: *Acroexch.AVDoc*

The *AVDoc* object represents the window containing the open PDF file. Your application can use this object for several purposes. It can have Acrobat render into the application's own window. This interface creates a window that closely resembles those that Acrobat

displays. It can also be used for selecting text, finding text, or printing pages. In addition, this object has several bridge methods to access other objects.

2.1.1.3 AVPageView

Apple event name: *AVPageView*

OLE automation name: *AcroExch.AVPageView*

The *AVPageView* object controls the contents of the *AVDoc* window, allowing your application to scroll, magnify, and go to the next, previous or arbitrary page. The history stack can be traversed through this object as well. This object, *AVDoc*, and *Application* are the primary objects by which you control the user interface.

2.1.1.4 AVMenu

Apple event name: *Menu*

OLE automation name: None

An *AVMenu* object represents a menu in the Acrobat viewer. You can count or remove menus. Each menu in the viewer has a language-independent name used to access it.

2.1.1.5 AVMenuItem

Apple event name: *Menu item*

OLE automation name: None

An *AVMenuItem* represents a single item in a menu. You can execute or remove menu items. Every menu item has a language-independent name used to access it.

2.1.2 PDF File Level

2.1.2.1 PDDoc

Apple event name: *Document*

OLE automation name: *AcroExch.PDDoc*

The *PDDoc* object is the hidden object behind every *AVDoc*. It is closest to the contents of a PDF file (at the IAC level). Through a *PDDoc*, an application can perform most of the Edit | Pages menu items from Acrobat (delete, replace, and so on.) Thumbnails can be created and deleted through this object. You can set and retrieve document information fields through this object as well.

Note: The first page in a document is page 1 for Apple events and page 0 for OLE automation.

2.1.2.2 PDPage

Apple event name: *PDPage*

OLE automation name: *AcroExch.PDPage*

Just as PDF files are partially composed of their pages, *PDDocs* are composed of *PDPages*. *PDPages* provide another opportunity for Acrobat to render into your application's window. You can also access page size and rotation through the *PDPage* object. Text selection regions can be set up. Annotations can be created and accessed through *PDPages*.

Note: The first page in a document is page 1 for Apple events and page 0 for OLE automation.

2.1.2.3 PDAnnot

Apple event name: *PDAnnot*

OLE automation name: *AcroExch.PDAnnot*

PDAnnots are the interface through which you can manipulate link and text annotations. Physical attributes of the annotation can be set and queried. The annotation can be performed when this makes sense: you can perform a link annotation but can't perform a text annotation.

Apple events have two additional, related objects: *PDTextAnnot*, a text annotation, and *PDLinkAnnot*, a link annotation.

2.1.2.4 PDBookmark

Apple event name: *PDBookmark*

OLE automation name: *AcroExch.PDBookmark*

PDBookmarks represent bookmarks in the PDF document. The *PDBookmark* object has somewhat limited use through the IAC interface. If you know the title of a bookmark, you can perform it, change its title, or delete the bookmark, but you can't directly create a bookmark.

2.1.2.5 PDTextSelect

Apple event name: None

OLE automation name: *AcroExch.PDTextSelect*

The *PDTextSelect* object serves two purposes. If selected text exists within an *AVDoc*, your application can access the words in that region through this object. If you wish to make text appear selected in an *AVDoc*, you create a *PDTextSelect*.

2.1.2.6 Hilite

Apple event name: None

OLE automation name: *AcroExch.Hilite*

A highlighted region of text in a PDF document. This object has a single method and is used by the *PDPage* object to create *PDTextSelect* objects.

2.2 Using Objects

The objects defined determine the kind of support Acrobat supplies an application.

2.2.1 Acrobat Viewer and PDF File Levels

On the Acrobat viewer level, you can display PDF files in your application's windows. You can do this in two ways, which differ in how much control you have over the application's user interface and appearance of its window. PDPAGE's *Draw* method merely gives you Acrobat's rendering capabilities. In addition to rendering, AVDoc's *OpenInWindow* method lets Acrobat do extra work for you. For example, if you use *OpenInWindow*, text annotations are displayed and links are active in your application's window—which is not the case if you use the *Draw* method.

On the PDF file level, you can do some basic manipulation of PDF documents, such as deleting, moving, or replacing pages; and changing annotation attributes. You can print PDF pages, select text, access manipulated text, and create or delete thumbnails.

2.2.2 Bridge Methods

At times it is necessary to get a certain object from related object of a different type. The API provides a set of *bridge methods* to acquire these objects.

For instance, you may want to get the PDDoc associated with an AVDoc. In OLE automation, you can do this with the *GetPDDoc* method in the AcroExch.AVDoc class.

Figure 3, “OLE Objects and Methods” in section 4.3 shows the OLE automation bridge methods graphically: arrows show how to get from one object to another.

2.2.3 Limitations

From the object discussion in the previous section, you could deduce that the capabilities just mentioned are available. It's less obvious that many of Acrobat's built-in tools and menu items are also accessible through the Application object. This opens up a world of possibilities, especially when you realize that tools and menu items added by plug-ins are also available through the Application object's methods.

Theoretically, this means that your application can do anything that a user can do with Acrobat, as if the application had "remote control" of the Acrobat viewer. However, see the Release Notes in the SDK for information on limitations.

Apple Event Support

Version 2.1, and later, of the Acrobat viewers on the Apple Macintosh support Apple events and a number of Apple event objects. The support includes some of the objects and events described in the *Apple Event Registry: Standard Suites*, as well as Acrobat-specific objects and events. This chapter lists each of the events and objects provided.

Apple events can be used from programming languages such as C or from AppleScript. Because AppleScript is much more straightforward, it is the recommended way for developers to use Apple events with the Acrobat viewers whenever possible. Some Apple events, such as the one that allows the Acrobat viewer to render into another application's window, cannot be accessed from AppleScript.

This chapter lists the Apple events supported by the Acrobat viewers. Information on Apple events supported by the Acrobat Search plug-in is contained in Technical Note #5162, [Acrobat Search API Reference](#). Other plug-ins can support additional Apple events, as described in Technical Note #5190, [Acrobat Core API Overview](#), which is part of the Acrobat Plug-ins SDK.

Note: When using Apple events, the first page of a document is page one, not page zero.

The constants needed to use the Apple events from a C program are included in header file *AETypes.h*, located in the IAC folder of the SDK.

For complete descriptions of the parameters associated with Apple events, see the Apple events sections of Technical Note #5165, [Acrobat Interapplication Communication Reference](#).

3.1 Differences Among the Acrobat Viewers

Acrobat supports all of the Apple events described in “[Apple event Suite Support](#)”.

Acrobat Reader supports only the four required Apple events: *run*, *open*, *print*, and *quit*.

3.2 Objects and Properties

The Acrobat viewer uses these standard objects and properties defined in the *Apple Event Registry: Standard Suites*:

- *cDocument* → *AVDoc* object, methods from *AVDoc* and *PDDoc*
- *cMenu* → *AVMenu* object and methods
- *cMenuItem* → *AVMenuItem* object and methods

Acrobat-specific objects retain their *PD/AV/IAS* prefix (*cAVPageView* and *cPDPage*, for example), but objects that are also present in an Apple standard object use the Apple name instead (*cApplication* and *cDocument*, for example).

The constants needed to use the Apple events from a C program are included in the header file *AETypes.h*, located in the IAC folder of the SDK.

Note: The prototype for *AEObjectInit* in the Apple-supplied *AEOObjects.h* is missing a void parameter, so *AETypes.h* contains a private, modified copy.

3.3 Acrobat Viewer Apple Event Objects

In addition to the standard objects mentioned above, the Acrobat viewer presents the following objects to the Apple event interface:

- *Application* — Represents the Acrobat viewer application itself.
- *Document* — Represents a single open document in the Acrobat viewer. The first page in a Document is page 1.
- *AVPageView* — Represents the view of the document in its window. Documents that aren't visible don't have AVPageViews.
- *PDPage* — Represents a single page in a PDF file. The first page in a Document is page 1.
- *PDAnnot* — An annotation on a page of a document. The PDTextAnnot and PDLinkAnnot classes are two specific annotation types.
- *PDTextAnnot* — A text annotation. Can only be used as the target of a *make* event. All other access is via the PDAnnot class.
- *PDLinkAnnot* — A link annotation. Can only be used as the target of a *make* event. All other access is via the PDAnnot class.
- *PDBookmark* — A bookmark.
- *Menu* — A menu in the Acrobat viewer.
- *Menu item* — A single item in a menu.

3.4 Apple event Suite Support

The Apple events supported by the Acrobat viewers are grouped into three categories:

- Required events — Events that the Finder sends to all applications.
- Core events — Events that are not universal to all applications, but are common to a wide variety of applications.
- Acrobat-specific events — Events that are specific to the Acrobat viewer.

This section describes each of the events in these categories.

For complete descriptions of the parameters associated with Apple events, see the Apple events sections of Technical Note #5165, [Acrobat Interapplication Communication Reference](#).

3.4.1 Required Suite

The required suite consists of four events the Finder sends to applications:

- *open* — Opens a file.
- *print* — Prints one or more files.
- *quit* — Terminates an application. See the quit event in the core suite for a variant that accepts options.
- *run* — Launches an application and invokes its standard startup procedures.

3.4.2 Core Suite

Acrobat viewers support this subset of the Core suite:

- *close* — Closes one or more documents.
- *count* — Counts the number of elements of a particular class in an object.
- *delete* — Deletes one or more objects.
- *exists* — Tests whether or not a specified object exists.
- *get* — Gets the value of a property of an object.
- *hide* — Hides the Acrobat viewer.
- *make* — Creates a new object.
- *move* — Moves a PDPAGE object.
- *open* — Opens a file into either a visible or a hidden window.
- *quit* — Terminates the Acrobat viewer.
- *save* — Saves a document to a file.
- *set* — Assigns one or more values to one or more variables.

3.4.3 Acrobat Viewer Suite

This suite contains Acrobat-specific events and objects. Apple encourages the use of an application's signature as the name of its class for application-specific Apple events, hence *CARO* is the name of the class for Acrobat viewer-specific Apple events. AppleScript users do not need to use this information.

```
#define kAEAcrobatViewerClass 'CARO'
```

- *bring to front* — Brings the specified document's window to the front.
- *clear selection* — Clears the document's current selection, if any.
- *close all docs* — Closes all documents.

- *create thumbs* — Creates thumbnail images for all pages in the document.
- *delete pages* — Deletes the specified pages in the document (the first page in a document is page 1).
- *delete thumbs* — Deletes all thumbnails from the document.
- *DrawPageToWindow* — Instructs the Acrobat viewer to render into a specified window instead of its own window.
- *execute* — Executes the specified menu item as if the user selected it.
- *find next note* — Finds and selects the next text note in a document.
- *find text* — Finds text in a document.
- *get info* — Gets the value of the specified key in the document's Info dictionary.
- *go backward* — Goes to the previous view in the stored view history.
- *go forward* — Goes to the next view in the stored view history.
- *goto* — Displays the page that has the specified page number.
- *goto next* — Displays the page after the one currently displayed in the AVPageView.
- *goto previous* — Displays the page before the one currently displayed in the AVPageView.
- *insert pages* — Inserts one or more pages from one document into another.
- *is toolbutton enabled* — Tests whether or not the specified button is enabled.
- *maximize* — Sets the document's window size to be the maximum or its original size.
- *perform* — Executes a bookmark's or link annotation's action.

- *print pages* — Prints one or more pages from a document, without displaying a modal Print dialog box.
- *read page down* — Scrolls forward through the document by one screen.
- *read page up* — Scrolls backward through the document by one screen.
- *remove toolbar* — Removes the specified button from the toolbar.
- *replace pages* — Replaces one or more pages in a document with pages from another document.
- *scroll* — Scrolls the view of a page by the specified amount.
- *select text* — Selects the specified text.
- *set info* — Sets the value of a specified key in the document's Info dictionary.
- *zoom* — Changes the zoom level of specified *AVPageView*.

CHAPTER 4

OLE Support

This chapter describes OLE 2.0 support in Adobe Acrobat for Microsoft Windows. Acrobat viewers are OLE servers and also respond to a variety of OLE automation messages. Since Acrobat is an OLE server, PDF documents can be embedded into documents created by an application that is an OLE client. The viewer's support for OLE automation allows it to respond to a variety of requests from other applications.

Developers who would like to improve their understanding of OLE automation can examine the AutoClik OLE automation Sample program and tutorial in Microsoft Visual C++ 1.5.X.

Note: The header files needed by C and C++ programmers to use the OLE automation support described in this chapter are located in the SDK IAC directory. Visual Basic users do not need these header files. However, these header files may be used to find the values of various constants.

For complete descriptions of the parameters associated with OLE automation methods, see the OLE automation sections of Technical Note #5165, [Acrobat Interapplication Communication Reference](#).

4.1 Differences Among the Acrobat Viewers

Acrobat supports all of the OLE automation methods listed in this chapter.

The Acrobat Reader does not support OLE automation.

4.2 OLE Server Support

The Acrobat viewer has the appropriate OLE interfaces to be an OLE server. PDF documents may be embedded or linked to in OLE containers.

Note: OLE server capability is added by the OLESRV16.API and OLESRV32.API plug-ins. These plug-ins work with the Acrobat Reader.

Acrobat OLE server support has the following limitations:

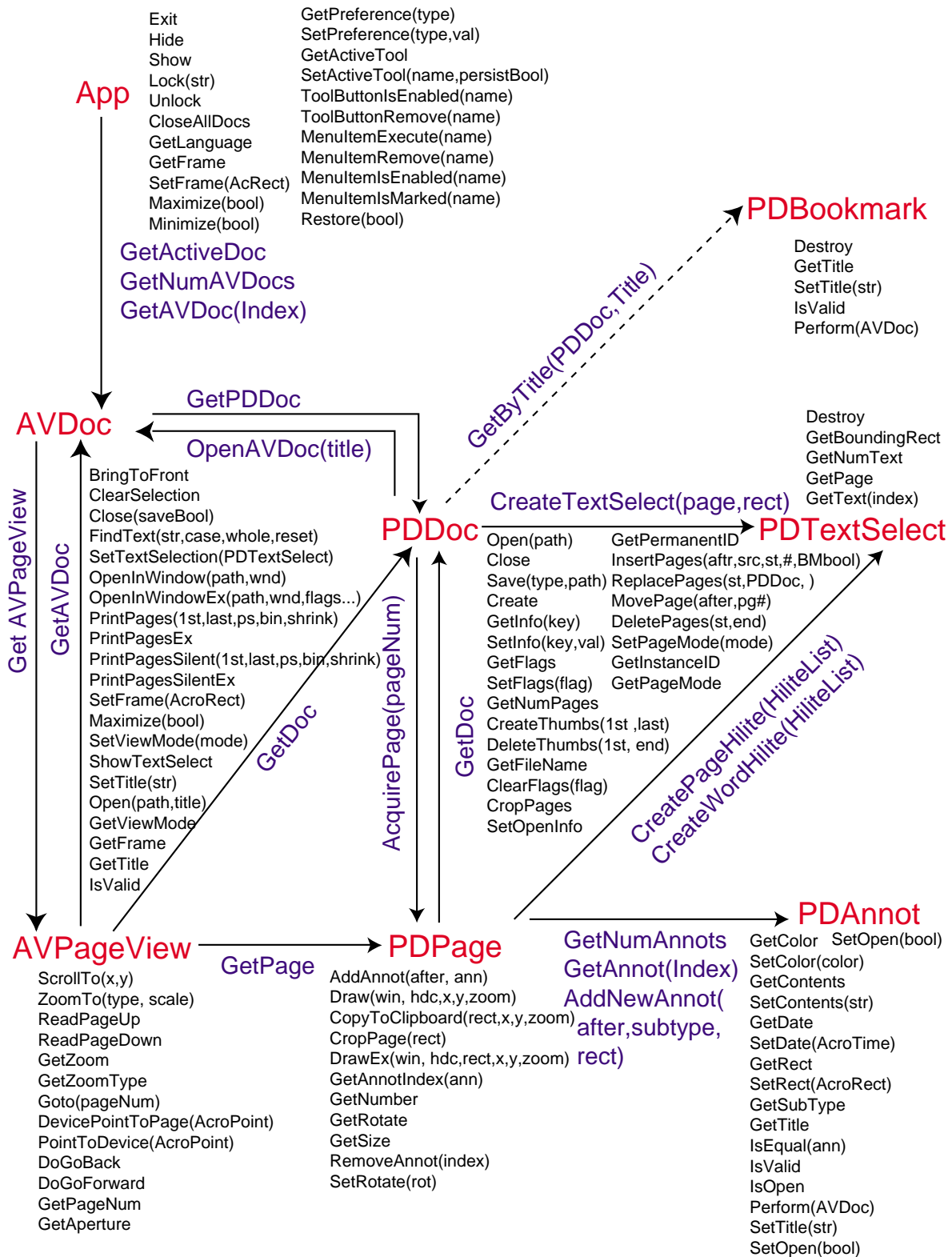
- Drag and drop is not supported.
- Acrobat does not do in-place activation.

4.3 OLE Automation Support

OLE automation support is provided by a set of classes. [Figure 3, “OLE Objects and Methods”](#) gives the layout of the classes and their methods. It also shows how to get from one object type to another via “bridge” methods.

Note: You can't use OLE automation with the Acrobat Reader.

Figure 3 OLE Objects and Methods



4.3.1 What's Possible: Examples and Approaches

For OLE automation, Acrobat exports primarily two capabilities: rendering PDF documents and remote control of the viewer. These tasks can be accomplished in a number of ways, which are presented below.

4.3.1.1 Rendering PDF Documents

PDF files can be rendered on the screen in two ways.

The first rendering method provides an interface similar to Acrobat's user interface. The *AVDoc* class's *OpenInWindow* or *OpenInWindowEx* method opens a PDF file in your application's window. This window has vertical and horizontal scroll bars, as well as buttons on the window's perimeter for setting the zoom factor. When the user interacts with this type of window, its operation is similar to working in Acrobat, though not everything works the same. For instance, links are active and the window displays any text annotations on a page.

The second means of rendering into your application's window uses the *PDPPage* class's *Draw* or *DrawEx* methods. With these methods, you provide a window and an *hDC* as well as a zoom factor. Acrobat renders the current page into that window. The application has to manage any scroll bars or anything else in the user interface. Acrobat merely renders the PDF file in that window. From a design standpoint, taking the *Draw* approach can work, unless your application requires an *AVDoc* representation.

The SDK provides two samples that illustrate these mechanisms. In the IAC Windows samples for MFC, the Visual C++ *DRAW* and *OIW* samples use the *Draw* and *OpenInWindow* methods respectively. The VB folder has parallel sample applications, *VBDRAW* and *VBOIW*.

4.3.1.2 Remotely Controlling the Viewer

There are a couple of ways to remotely control Acrobat. Given the exported interfaces, you can write an application that manipulates PDF files. Various aspects of PDF files can be manipulated, such as pages, annotations, and bookmarks. Such an application might use AVDoc, PDDoc, PDPage and PDAnnot methods. It might not provide any visual feedback that required rendering into its application window.

Some have deployed Acrobat as a help system, launching Acrobat from their own application, which has set up an environment for the user. The application has Acrobat open some file, set the page location, zoom factor, and possibly even select some text.

4.3.2 Development Environments

Developers have a choice of environments in which to integrate with Acrobat: Visual Basic and Visual C++.

The preferred environment is Visual Basic. The run-time typechecking offered by the *CreateObject* call in Visual Basic allows quick prototyping of an application. When you build a similar application with C or C++, typechecking is done at compile time.

Consider the following Basic subroutine to view a given page:

```
Sub Goto(Int where)
Dim app as Object, avdoc as Object, pageview
as Object

Set app = CreateObject("AcroExch.App")
Set avdoc = app.GetActiveDoc
Set pageview = avdoc.GetAVPageView
pageview.Goto(where)
End Sub
```

The routine looks like this in Visual C++:

```
void goto(int where)
{
    CAcroApp app;
    CAcroAVDoc *avdoc = new CAcroAVDoc;
    CAcroAVPageView pageview;
    COleException e;
    app.CreateDispatch("AcroExch.App", &e);
    avdoc->AttachDispatch(app.GetActiveDoc,
    TRUE);
    pageview->AttachDispatch(avdoc->GetAVPageView
    , TRUE);
    pageview->Goto(where);
}
```

Arguably, the Visual Basic example is simpler to read, write, and support. In Visual C++, the *CAcro...* classes hide much of the typechecking that must be done. Using OLE automation objects in Visual C++ requires understanding the *AttachDispatch* and *CreateDispatch* methods of the *COleDispatchDriver* class. See [“Using Visual Basic with the Acrobat SDK”](#) for more information.

4.3.3 Using Visual Basic with the Acrobat SDK

The Microsoft Windows versions of Acrobat and Acrobat Capture 2.0, or later, software are OLE servers. The only requirement for using the OLE objects made available by Acrobat and Capture is to have the product installed on your system and the appropriate *.tlb* file included in the project references for your Visual Basic project. The Acrobat *.tlb* file is named *Acrobat.tlb* and the Capture file is called *capserve.tlb*. These files are included in the IAC *headers.zip* file located on the Adobe public Web site. Once you have the *.tlb* file included in your project, you will be able to use the Visual Basic object browser to browse the OLE objects. It is not sufficient to install just an ActiveX control or DLL to enable OLE Automation. You must have the full product (Acrobat or Capture) installed to use OLE automation.

4.3.3.1 Getting Started

This technical note and Technical Note #5165, [Acrobat Interapplication Communication Reference](#), document the objects and methods available. These documents (as well as the API) were designed with C programming in mind and programming with the API requires some familiarity with C concepts such as “enums.”

The best resources for a VB programmer, besides the object browser, are the sample projects VBOIW and VBDraw. These samples demonstrate use of the Acrobat OLE objects and contain comments describing the parameters for the more complicated methods.

4.3.3.2 What Must I know About C to Use Visual Basic with Acrobat Software?

Downloading the IAC.h (included in the IAC [headers.zip](#) file) will be helpful for a VB programmer. Some of the methods such as *OpenInWindowEx* can be confusing when utilized in VB. The method is prototyped as taking a long for the *openflags* parameter. The options for this parameter provided in the IAC Reference are:

- *AV_EXTERNAL_VIEW* — Open the document with the toolbar visible.
- *AV_DOC_VIEW* — Draw the page pane and scrollbars.
- *AV_PAGE_VIEW* — Draw only the page pane.

If you were developing in the C Programming Language, these strings would be replaced by a numeric value prior to compilation; passing these strings to the method would not raise an error. When programming in VB, these strings are not replaced so you must look in the IAC.h header file to determine the appropriate numeric value. In IAC.h, you will find the following enum.

```

typedef enum _t_AVViewType {
AV_EXTERNAL_VIEW = 1, /* Open the document
with toolbar visible */
AV_DOC_VIEW = 2,      /* Draw the page pane
and scrollbars */
AV_PAGE_VIEW = 4      /* Draw only the page
pane */
} AVViewType;

```

For example, if you wanted to draw only the PDF page, you would pass 4 for the *openflags* parameter.

In some situations, you will need to do a bitwise OR of multiple values and pass it to a method. An example of this is using *PDDocSave*. The *ntype* parameter of the *PDDocSave* method is a bitwise OR of the following flags as defined in *IAC.h*:

```

/* PDSaveFlags – used for PD-level Save
** All undefined flags should be set to zero.
** If either PDSaveCollectGarbage or PDSaveCopy
are used, PDSaveFull must be used. */
typedef enum {
PDSaveIncremental = 0x0000, /* write changes
only */
PDSaveFull = 0x0001,        /* write entire
file */
PDSaveCopy = 0x0002,        /* write copy w/
o affecting current state */
PDSaveLinearized = 0x0004,  /* writes the
file linearized */
PDSaveCollectGarbage = 0x0020 /* perform
garbage collection on unreferenced objects */
} PDSaveFlags;

```

If you want to fully save the PDF file and linearize it, you would pass 5 for the *ntype* parameter, since that is the binary OR of the *PDSaveFull* and *PDSaveLinearized* parameters. In many instances, the numeric values are spelled out in comments in the VB sample code, however, knowledge of why the methods are

structured in this way and how the method would be utilized in C can help VB programmers when these values are not spelled out clearly.

4.3.4 Understanding the MFC Interfaces to OLE

4.3.4.1 Background

OLE 2.0 support includes several *CAcro...* classes (*CAcroApp* and *CAcroPDDoc*, for instance) to simplify driving Acrobat. Several files in the SDK encapsulate the definitions of these classes.

The *CAcro...* classes are defined in Object Description Language in *ACROBAT.ODL*. The *MkTypLib* utility produces the type library *ACROBAT.TLB* from *ACROBAT.ODL*. The *OLE2View* tool in the OLE 2.0 SDK allows you to browse registered type libraries and instantiate objects to list their interfaces. Use *ACROBAT.TLB* when defining OLE automation for a project in Microsoft Visual C++. In the process, Class Wizard generates the files *ACROBAT.H* and *ACROBAT.CPP*, which implement a type-safe wrapper to the Acrobat automation server. The Acrobat SDK already includes the *ACROBAT.H* and *ACROBAT.CPP* files, so you don't have to generate them yourself.

Note: *The ACROBAT.ODL, ACROBAT.TLB, ACROBAT.H, and ACROBAT.CPP files in the SDK should not be modified: these define Acrobat's OLE automation interface.*

The *CAcro...* classes inherit from the MFC *COleDispatchDriver* class. Understanding this class makes it easier to write applications that use the *CAcro...* classes and their methods.

The *COleDispatchDriver* class implements the client side of OLE automation, providing most of the code needed to access automation objects. It provides several wrapper functions: *AttachDispatch*, *DetachDispatch*, and *ReleaseDispatch*, as well as several convenience functions: *InvokeHelper*, *SetProperty*, and *GetProperty*. You employ

some of these methods when you use the Acrobat-provided automation objects. Other methods are used in Acrobat's implementation of these objects.

See Technical Note #5165, [Acrobat Interapplication Communication Reference](#), for details on the *CAcro...* classes and their methods.

4.3.4.2 Using the *COleDispatchDriver* Class With Acrobat

This section discusses how to use the classes exported by *ACROBAT.CPP*. It shows when to call the *CreateDispatch* and *AttachDispatch* methods. The section explains in detail a code sample that uses these methods in the *COleDispatchDriver* class.

Consider this implementation of a method adapted from the OIW sample in the SDK:

```
/* OnToolsHiliteWords is called when the user
clicks on a toolbutton. It highlights
(creates a text selection of) the first 10
words on the current page.
*/
void COiwDoc::OnToolsHiliteWords
{
    COLEException e;
    CAcroAVPageView tpv;
    CAcroHiliteList *hilite = new
    CAcroHiliteList;
    CAcroPDPage page;

    // Obtain the AVPageView
    tpv.AttachDispatch(m_pAcroAVDoc->GetAVPageVie
w, TRUE);

    // Create the hilite list object
    hilite->CreateDispatch("AcroExch.HiliteList",
&e);
    if (hilite) {
        // highlight first 10 words
```

```

        hilite->Add(0, 10);
        page.AttachDispatch(tpv.GetPage, TRUE);
//m_pAcroAVDoc contains pointer to active
AVDoc
        m_pAcroAVDoc->SetTextSelection

(page.CreateWordHilite(hilite->m_lpDispatch))
;
        m_pAcroAVDoc->ShowTextSelect;
    }
}

```

This C++ code may appear cryptic if you're unfamiliar with the *COleDispatchDriver* class. There is little printed documentation on this class; the on-line Windows Help documentation has some information.

However, you don't need to know the details of this class to use it effectively with the Acrobat viewer. Most details of OLE automation are hidden from the user by ClassWizard and MFC, so you can concentrate on implementing your functions with the Acrobat viewer. *COleDispatchDriver* is generally used only by classes created by ClassWizard, such as the *CAcro...* classes. When you create new C++ classes by using a type library such as *ACROBAT.TLB*, ClassWizard derives the classes from *COleDispatchDriver*.

Member functions of *COleDispatchDriver* handle a dispatch connection. *IDispatch* is the OLE interface by which applications expose methods and properties so that other applications such as Visual BASIC can use the application's features. This provides Visual BASIC OLE support for Acrobat viewers. *COleDispatchDriver* is essentially a "class wrapper" for *IDispatch*. An *LPDISPATCH* is a *long* pointer to an *IDispatch*, which can be considered an opaque data type representing a dispatch connection.

In the above sample, the objects with the prefix *CAcro-* are all *CAcro...* class objects—and they are also *COleDispatchDriver* objects—since all the Acrobat *CAcro...* classes are subclasses of *COleDispatchDriver*.

You can find a complete list of *CAcro...* classes and their methods in the SDK's *ACROBAT.H* file. Here is a section of code from *ACROBAT.H* that declares the *CAcroHiliteList* class:

```
class CAcroHiliteList : public
COleDispatchDriver
{
// Attributes
public:
//Operations
public:
    BOOL Add(short nOffset, short nLength);
}
```

Here is the related implementation section from *ACROBAT.CPP*:

```
// CAcroHiliteList operations
BOOL CAcroHiliteList::Add(short nOffset,
short nLength)
{
    BOOL result;
    static BYTE BASED_CODE parms[] = VTS_I2
VTS_I2;
    InvokeHelper(0x1, DISPATCH_METHOD, VT_BOOL,
        (void *)&result, parms, nOffset, nLength);
    return result;
}
```

The *CAcroHiliteList* class is a *COleDispatchDriver* class with one additional method—*Add*. This means that it shares all the instance variables of the *COleDispatchDriver* class. One of these variables is *m_lpDispatch*, which holds an *LPDISPATCH* for that object.

The *m_lpDispatch* variable can be used in functions that require an *LPDISPATCH* argument. For instance, the *InsertPages* method takes an *LPDISPATCH* argument for the source document of the pages to insert. If *sourcePDDoc* points to the *CAcroPDDoc* that is the source document, use *sourcePDDoc->m_lpDispatch* for the *LPDISPATCH* argument in the call to *InsertPages*.

When this line is executed

```
hilite->Add(0, 10);
```

the *InvokeHelper* function gets called. This *COleDispatchDriver* method takes a variable number of arguments. It eventually calls the Acrobat implementation for *CAcroHiliteList's* *Add* method. This happens across the virtual OLE “wires” and takes care of all the OLE details. The end result is that a page range is added to the *CAcroHiliteList* object.

Instantiating a class isn't sufficient to use it. Before you use an object, you must *attach* to the appropriate Acrobat object. This is done by using one of the *-Dispatch* methods of the *COleDispatchDriver* class. These functions also initialize the *m_lpDispatch* instance variable for the object.

The following example illustrates attaching an *IDispatch* that already exists:

```
CAcroAVPageView tpv;  
// Get the AVPageView  
tpv.AttachDispatch(m_pAcroAVDoc->GetAVPageView,  
w, TRUE);
```

The *GetAVPageView* method of the *CAcroAVDoc* class returns an *LPDISPATCH*—which is what the *AttachDispatch* method is expecting for its first argument. The *BOOL* passed as the second argument indicates whether or not the *IDispatch* should be released when the object goes out of scope and is typically *TRUE*. In general, when an *LPDISPATCH* is returned from a method such as *GetAVPageView*, you use *AttachDispatch* to attach it to an object.

Here's an example using the *CreateDispatch* method:

```
CAcroHiliteList *hilite = new
CAcroHiliteList;

/* Create the hilite list object */
hilite->CreateDispatch("AcroExch.HiliteList",
&e);

hilite->Add(0, 10);
```

In this case, the *CreateDispatch* method both creates the *IDispatch* object and attaches it to the object. This code works fine; however, if the code had been

```
CAcroHiliteList *hilite = new
CAcroHiliteList;
hilite->Add(0, 10);
```

the application would fail. This error is analogous to using an uninitialized variable. Until the *IDispatch* object is attached to the *COleDispatchDriver* object, it isn't valid.

CreateDispatch has two forms, the first of which takes a *REFCLSID* argument. The second form takes a string, such as "AcroExch.HiliteList". Getting this string wrong is a common mistake. Here's an example of using the wrong string:

```
CAcroPDDoc doc = new CAcroPDDoc;
doc.CreateDispatch("AcroExch.Create", &e);
```

This fails because Acrobat won't respond to such a "verb". It should be "AcroExch.PDDoc" instead. [Table 1, "Strings for CreateDispatch"](#) lists all the valid strings.

Now let's look more closely at the example above. The variable *tpv* refers to a page view. In order to use *tpv*, declare it and attach a dispatch to it. This code piece illustrates that:

```

CAcroAVPageView tpv;
// Get the AVPageView
tpv.AttachDispatch(m_pAcroAVDoc->GetAVPageView, TRUE);

```

Now let's work on the page:

```

CAcroPDPPage page;
page.AttachDispatch(tpv.GetPage, TRUE);
m_pAcroAVDoc->SetTextSelection
(page.CreateWordHilite(hilite->m_lpDispatch))
;
m_pAcroAVDoc->ShowTextSelect;

```

The first line allocates space for the *CAcroPDPPage* variable *page*. Since it's a *CAcro*- variable, we need to attach to the OLE object before we can use its methods. We know we can't use *CreateDispatch* to create a *PDPPage* since there's no "AcroExch.PDPPage" listed in [Table 1, "Strings for CreateDispatch"](#). Therefore, we should look for a class with a method that returns an *LPDISPATCH* for a *PDPPage*. Experienced plug-in developers know you can get a *PDPPage* from a *PDDoc*, but [Figure 3, "OLE Objects and Methods"](#) also provides a hint of where to look. In this case, we can get a *PDPPage* from the *AVPageView* method *GetPage*. We need a *PDPPage* because the purpose of this code is to highlight words on the current page, which is represented by the *AVPageView* object.

The *GetPage* method of the *AVPageView* object returns an *LPDISPATCH*. This is passed as the first argument to the *AttachDispatch* method of the page object. The *TRUE* argument indicates we want the object auto-released when it goes out of scope.

The third line is the most complicated:

```

m_pAcroAVDoc->SetTextSelection
(page.CreateWordHilite(hilite->m_lpDispatch))
;

```

Let's work from the outside in. The goal of the function is to select the first ten words on the page. To do that, it needs to set a text selection for the current document.

The *CAcroAVDoc* object has a *SetTextSelection* method, which takes a *PDTextSelect* object argument. The instance variable *m_pAcroAVDoc* contains an *LPDispatch* for the current active AVDoc (this is set up for us by the OIW application from which this code is excerpted). So this line performs the *SetTextSelection* method on the current AVDoc. The *CAcroPDPPage* class has a method that returns an *LPDISPATCH* for a *PDTextSelect*: *CreateWordHilite*. (We could also use *CreatePageHilite*.) The *CreateWordHilite* method takes an *LPDISPATCH* argument representing a *CAcroHilite* list. We've already built a *CAcroHiliteList* object—it's in the *hilite* variable. The *COleDispatchDriver* class has an instance variable *m_lpDispatch* that contains the *LPDISPATCH* for the object. The result of calling *SetTextSelection* is the selection of the first ten words on the current page.

Finally, to cause the visual update on the screen, we call the *AcroAVDoc*'s *ShowTextSelect* method.

Table 1: Strings for CreateDispatch

CAcroPoint: "AcroExch.Point"

CAcroRect: "AcroExch.Rect"

CAcroTime: "AcroExch.Time"

CAcroApp: "AcroExch.App"

CAcroPDDoc: "AcroExch.PDDoc"

CAcroAVDoc: "AcroExch.AVDoc"

CAcroHiliteList: "AcroExch.HiliteList"

CAcroPDBookmark: "AcroExch.PDBookmark"

CAcroMatrix: "AcroExch.Matrix"

4.4 Development Information

4.4.1 Using OLE Messages

The Acrobat OLE automation implementation is based on a synchronous messaging scheme. This requires some care in sending messages.

An application sends a request to the Acrobat viewer, it processes that request, and returns control to the application. Only then can the application send Acrobat another message. If it sends one message, then another immediately afterward, the second message is not sent properly. (This does not generate a server busy error, but fails with no error message.)

This problem usually manifests itself with the *AVDoc.OpenInWindow* method—where a large amount of traffic goes back and forth regarding drawing position and mouse clicks—and the *PDPPage.Draw* method—especially on complex pages. With the *Draw* method, the problem arises when a *WM_DRAW* message is generated, which calls the viewer to draw the page. If the page is complex and the environment is multi-threaded, the viewer may not finish drawing the page before the application generates another *WM_DRAW* message. Another draw message is then sent to the viewer, which causes the problem—since the viewer is single-threaded.

Though this problem primarily occurs with the *OpenInWindow* or *Draw* methods, it can occur with any Acrobat OLE automation method. User input through keyboard events or mouse clicks can cause this problem, since those events can also cause OLE messages to be sent to the viewer.

A partial solution is to queue all OLE messages for Acrobat and send them to Acrobat at a metered rate. This works in every case except using *OpenInWindow* and *Draw*.

Many OLE automation problems can be alleviated by increasing swap space to 20 MB or greater.

4.4.2 MDI Usage

Suppose you create a multiple document interface application that creates a static window into which Acrobat displays (using the *OpenInWindowEx* call), and this window is based on the *CFormView* OLE class. If another window is placed on top of that window then removed, the Acrobat window does not repaint correctly.

To fix this, make the dialog template, on which *CFormView* is based, have the Clip Children style. Otherwise, the dialog erases the background of all child windows, including the one containing the PDF file, which wipes out the previously covered part of the PDF window.

4.4.3 Event Handling

When a PDF file is opened with *OpenInWindow/Ex*, the Acrobat viewer creates a child window on top of it. This allows the viewer to receive events for this window directly. However, an application also has to do some event handling for the following events: resize, key up, and key down.

The following function from the OIW sample shows how to handle a resize event:

```
void COiwView::OnSize(UINT nType, int cx, int cy)
{
    CWnd* pWndChild = GetWindow(GW_CHILD);
    if (!pWndChild)
        return;
    CRect rect;
    GetClientRect(&rect);
    pWndChild->
        SetWindowPos(NULL, 0, 0, rect.Width, rect.Height,
        SWP_NOZORDER | SWP_NOMOVE);

    CView::OnSize(nType, cx, cy);
}
```

After sending the message to the child window, it also does a resize. This results in both windows being resized, which is the desired effect.

4.4.4 Using OLE Automation With Visual Basic

You can use the OLE automation classes with Visual Basic.

4.4.4.1 Constant Values

Although you do not need the header files provided in the SDK, these files may be used to find the values of various constants, such as *AV_DOC_VIEW*, that are referenced in the documentation. The file *IAC.H* contains most of these values.

4.4.4.2 Acrobat 3.0 Changes

The following changes are required with Visual Basic applications in a 32-bit environment:

- Use the Visual Basic 5.0 Project->References->Browse menu item to add *Acrobat.tlb*, the Acrobat type library, to your project. *Acrobat.tlb* resides in the *headers* directory. Before running the project, make sure it is saved.
- Instead of defining Acrobat Objects as Objects, define them as the desired class. For example, change

```
Dim pdDoc As Object
```

to

```
Dim pdDoc As CAcroPDDoc
```

4.5 OLE Automation Summary

This section simply lists all the OLE automation methods. For complete descriptions of the parameters associated with OLE automation methods, see the OLE automation sections of Technical Note #5165, [Acrobat Interapplication Communication Reference](#).

4.5.1 Objects

The Acrobat viewer is represented as several OLE automation C++ objects:

- *AcroExch.App* — The application itself.
- *AcroExch.AVDoc* — A document as seen in the user interface.
- *AcroExch.PDDoc* — The underlying PDF representation of a document. The first page in a PDDoc is page 0.
- *AcroExch.AVPageView* — The window pane in which the document is drawn.
- *AcroExch.PDPage* — A single page in the PDF representation of a document. The first page in a PDDoc is page 0.
- *AcroExch.PDAnnot* — An annotation on a page in the PDF file.
- *AcroExch.PDBookmark* — A bookmark in a PDF file.
- *AcroExch.PDTextSelect* — A selection of text in the Acrobat viewer, as if a user had used the mouse to select a region of text.

4.5.2 Data Types

Acrobat supports the following data types for OLE automation:

- *AcroExch.Hilite* — An entry in a highlight list. A highlight list is used to highlight one or more groups of characters/words on a single page.
- *AcroExch.HiliteList* — A list of highlighted characters, which may include one or more contiguous groups of characters or words on a single page. The *Add* function is provided to add to a *HiliteList*.
- *AcroExch.Point* — A point, specified by its *x*- and *y*-coordinates.

- *AcroExch.Rect* — A rectangle, specified by the top left and bottom right points.
- *AcroExch.Time* — A specified time, accurate to the millisecond, including the day of the week.

4.5.3 Methods

Acrobat OLE automation support includes the following C++ methods, grouped by object.

4.5.3.1 AcroExch.App

- *CloseAllDocs* — Closes all open documents.
- *Exit* — Exits the Acrobat viewer.
- *GetActiveDoc* — Gets the frontmost document.
- *GetActiveTool* — Gets the name of the currently active tool.
- *GetAVDoc* — Gets an AVDoc by its index in the list of open AVDocs.
- *GetFrame* — Gets the window's frame.
- *GetLanguage* — Gets a code that specifies which language the Acrobat viewer's user interface is using.
- *GetNumAVDocs* — Gets the number of open AVDocs.
- *GetPreference* — Gets a value in the preferences file for zoom values and colors.
- *Hide* — Hides the Acrobat viewer.
- *Lock* — Locks the Acrobat viewer.
- *Maximize* — Maximizes the Acrobat viewer.
- *MenuItemExecute* — Executes the menu item whose language-independent menu item name is specified.
- *MenuItemIsEnabled* — Determines if the specified menu item is enabled or not.
- *MenuItemIsMarked* — Determines if the specified menu item marked or not.

- *MenuItemRemove* — Removes the menu item whose language-independent menu item is specified.
- *Minimize* — Minimizes the Acrobat viewer.
- *Restore* — Restores the Acrobat viewer window to its previous state.
- *SetActiveTool* — Sets the active tool to the tool whose name is specified.
- *SetFrame* — Sets the window's frame to the specified rectangle.
- *SetPreference* — Sets a value in the preferences file for zoom values and colors.
- *Show* — Shows the Acrobat viewer.
- *ToolButtonsEnabled* — Determines if the specified toolbar button is enabled or not.
- *ToolButtonRemove* — Removes the specified button from the toolbar.
- *Unlock* — Unlocks the Acrobat viewer if it was previously locked.

4.5.3.2 AcroExch.AVDoc

- *BringToFront* — Brings the window to the front.
- *ClearSelection* — Clears the current selection.
- *Close* — Closes a document.
- *FindText* — Finds the specified text, scrolls so that it is visible, and highlights it.
- *GetAVPageView* — Gets the AVPageView associated with an AVDoc.
- *GetFrame* — Gets the rectangle specifying the window's size and location.
- *GetPDDoc* — Gets the PDDoc associated with an AVDoc.
- *GetTitle* — Gets the window's title.

- *GetViewMode* — Gets the current document view mode (pages only, pages and thumbnails, or pages and bookmarks).
- *IsValid* — Determines whether the AVDoc is still valid.
- *Maximize* — Maximizes the window if MaxSize is true.
- *Open* — Opens a file.
- *OpenInWindow* — Opens a PDF file and displays it in a user-specified window.
- *OpenInWindowEx* — (New in Acrobat 3.0) Opens a PDF file and displays it in a user-specified window, specifying a page number, zoom factor, and page view.
- *PrintPages* — Prints a specified range of pages, displaying a print dialog box.
- *PrintPagesSilent* — (New in Acrobat 3.0) Prints a specified range of pages without displaying any print dialog box.
- *SetFrame* — Sets the window's size and location.
- *SetTextSelection* — Sets the document's selection to the specified, previously created text selection.
- *SetTitle* — Sets the window's title.
- *SetViewMode* — Sets the mode in which the document is viewed (pages only, pages and thumbnails, or pages and bookmarks).
- *ShowTextSelect* — Changes the view so that the current text selection is visible.

4.5.3.3 AcroExch.AVPageView

- *DevicePointToPage* — Converts the coordinates of a point from device space to user space.

Note: Do not use the *DevicePointToPage* method: it will not be available in future versions of Acrobat. Furthermore, there is no way to translate between device space and machine space.

- *DoGoBack* — Goes to the previous view on the view history stack, if any.
- *DoGoForward* — Goes to the next view on the view history stack, if any.
- *GetDoc* — Gets the PDDoc corresponding to the current page.
- *GetAperture* — (*New in Acrobat 3.0*) Gets the aperture associated with the current page. The *aperture* is the rectangular region of the window in which the document is drawn, measured in device space units.
- *GetDoc* — Gets the PDDoc corresponding to the current page.
- *GetPage* — Gets the PDPage corresponding to the current page.
- *GetPageNum* — Gets the page number of the page. The first page in a document is page zero.
- *GetZoom* — Gets the current zoom factor, specified as a percent, for example, 100 is returned if the magnification is 1.0.
- *GetZoomType* — Gets the current zoom type.
- *Goto* — Goes to the specified page.
- *PointToDevice* — Converts the coordinates of a point from user space to device space.

Note: Do not use the PointToDevice method: it will not be available in future versions of Acrobat. Furthermore, there is no way to translate between device space and machine space.

- *ReadPageDown* — Scrolls forward through the document by "one screenfull."
- *ReadPageUp* — Scrolls backward through the document by "one screenfull."
- *ScrollTo* — Scrolls to the specified location on the current page.
- *ZoomTo* — Zooms to a specified magnification.

4.5.3.4 AcroExch.HiliteList

- *Add* — Adds a specified highlight to the current highlight list.

4.5.3.5 AcroExch.PDAnnot

- *GetColor* — Gets an annotation's color.
- *GetContents* — Gets a text annotation's contents.
- *GetDate* — Gets an annotation's date.
- *GetRect* — Gets an annotation's bounding rectangle.
- *GetSubtype* — Gets an annotation's subtype.
- *GetTitle* — Gets a text annotation's title.
- *IsEqual* — Determines whether or not an annotation is the same as a specified annotation.
- *IsOpen* — Tests whether or not a text annotation is open.
- *IsValid* — Tests whether or not an annotation is still valid.
- *Perform* — Performs a link annotation's action.
- *SetColor* — Sets an annotation's color.
- *SetContents* — Sets a text annotation's contents.
- *SetDate* — Sets an annotation's date.
- *SetOpen* — Opens or closes a text annotation.
- *SetRect* — Sets an annotation's bounding rectangle.
- *SetTitle* — Sets a text annotation's title.

4.5.3.6 AcroExch.PDBookmark

Note: It is not possible to create a bookmark with OLE—only to destroy one.

- *Destroy* — Destroys a bookmark. It is not possible to create a bookmark with OLE.

- *GetByTitle* — Gets the bookmark that has a specified title.
- *GetTitle* — Gets a bookmark's title (up to 256 characters).
- *IsValid* — Tests whether or not a bookmark is still valid.
- *Perform* — Performs a bookmark's action.
- *SetTitle* — Sets a bookmark's title.

4.5.3.7 AcroExch.PDDoc

- *AcquirePage* — Acquires the specified page.
- *ClearFlags* — Clears a document's flags. The flags indicate whether the document has been modified, whether the document is a temporary document and should be deleted when closed, and the version of PDF used in the file. This method can be used only to clear, not to set, flag bits. Not available prior to Acrobat 2.1.
- *Close* — Closes a file.
- *Create* — Creates a new PDDoc.
- *CreateTextSelect* — Creates a text selection from the specified rectangle on the specified page.
- *CreateThumbs* — Creates thumbnail images for the specified page range in a document.
- *CropPages* — Crops the pages in a specified page range.
- *DeletePages* — Deletes pages from a file.
- *DeleteThumbs* — Deletes thumbnail images from the specified pages in a document.
- *GetFileName* — Gets the name of the file associated with this PDDoc.
- *GetFlags* — Gets a document's flags. The flags indicate whether the document has been modified, whether the document is a temporary document and should be deleted when closed, and the version of PDF used in the file.

- *GetInfo* — Gets the value of a specified key in the document's info dictionary.
- *GetInstanceID* — Gets the instance ID from the ID array in the document's trailer.
- *GetNumPages* — Gets the number of pages in a file.
- *GetPageMode* — Gets a value indicating whether the Acrobat viewer is currently displaying only pages, pages and thumbnails, or pages and bookmarks.
- *GetPermanentID* — Gets the permanent ID from the ID array in the document's trailer.
- *InsertPages* — Inserts pages into a file.
- *MovePage* — Moves a page to another location within the same document.
- *Open* — Opens a file.
- *OpenAVDoc* — Opens a window and displays the document in it.
- *ReplacePages* — Replaces pages in a file with those from a specified file.
- *Save* — Saves a document.
- *SetFlags* — Sets a document's flags. The flags indicate whether the document has been modified, whether the document is a temporary document and should be deleted when closed, and the version of PDF used in the file. This method can be used only to set, not clear, flag bits.
- *SetInfo* — Sets the value of a key in a document's info dictionary.
- *SetOpenInfo* — Sets the parameters that tell how a document is opened.
- *SetPageMode* — Sets the page mode in which a document is opened: display only pages, pages and thumbnails, or pages and bookmarks.

4.5.3.8 AcroExch.PDPage

- *AddAnnot* — Adds a specified annotation at a specified location in the page's annotation array.
- *AddNewAnnot* — Creates a new text annotation and adds it to the page.
- *CopyToClipboard* — (New in Acrobat 3.01, available only on 32-bit systems) Copies a PDF image to the clipboard without requiring an *hWnd* or *hDC*.
- *CreatePageHilite* — Creates a text selection from a list of character offsets and character counts on a single page.
- *CreateWordHilite* — Creates a text selection from a list of word offsets and word counts on a single page.
- *CropPage* — Crops a single page.
- *Draw* — Instructs the Acrobat viewer to draw into a specified window.
- *DrawEx* — (New in Acrobat 3.0) Instructs the Acrobat viewer to draw into a specified window, specifying a page view and zoom factor.
- *GetAnnot* — Gets the n^{th} Index annotation on the page.
- *GetAnnotIndex* — Gets the index (in the page's annotation array) of the specified annotation.
- *GetDoc* — Gets the PDDoc associated with the page.
- *GetNumAnnots* — Gets the number of annotations on the page.
- *GetNumber* — Gets the page number of the current page. The first page in a document is page zero.
- *GetRotate* — Gets the rotation value for the current page.
- *GetSize* — Gets a page's width and height.
- *RemoveAnnot* — Removes the specified annotation from the page's annotation array.

- *SetRotate* — Sets the rotation for the current page.

4.5.3.9 AcroExch.PDTextSelect

- *Destroy* — Destroys a text selection.
- *GetBoundingRect* — Gets a text selection's bounding rectangle.
- *GetNumText* — Gets the number of text elements in a text selection.
- *GetPage* — Gets the page number on which a text selection is located.
- *GetText* — Gets the text from the specified element of a text selection.

CHAPTER 5

DDE Support

This chapter describes DDE support in the Acrobat viewers under Microsoft Windows. Available DDE messages are listed. Developers are encouraged to use OLE automation instead of DDE, whenever possible.

This chapter lists all DDE messages. For complete descriptions of the parameters associated with DDE messages, see the DDE sections of Technical Note #5165, [Acrobat Interapplication Communication Reference](#).

5.1 Differences Among the Acrobat Viewers

Acrobat 4.0 supports 32-bit applications.

Acrobat supports all of the DDE messages listed in “[Acrobat Viewer DDE Messages](#)”.

Acrobat Reader supports only the following DDE messages: *AppExit*, *CloseAllDocs*, *DocClose*, *DocGoTo*, *DocGoToNameDest*, *DocOpen*, *FileOpen*, *FilePrint*, *FilePrintEx*, *FilePrintSilent*, and *FilePrintTo*.

5.2 General Information

Here is information needed to use DDE messages:

- For all DDE messages listed in this chapter, the service name is *acroview*, the transaction type is *XTYPE_EXECUTE*, and the topic name is *control*. The data is the command you wish to execute, enclosed by square brackets. The item argument in the *DdeClientTransaction* call is *NULL*.

The following example sets up a DDE message:

```
DDE_SERVERNAME = "acroview";  
DDE_TOPICNAME = "control";  
DDE_ITEMNAME = "[AppHide()]";
```

Note: The square bracket characters [and] in DDE messages are significant, and must be included as part of the message.

Note: DDE messages are case-sensitive and must be used exactly as described.

- You must first open a document using the *DocOpen* DDE message in order to be able to use other DDE messages on it. You cannot use DDE messages to close a document that a user opened manually, for example.
- Where a pathname is used, it may be *NULL*, in which case the DDE message operates on the front document.
- If more than one command is sent at once, they are executed sequentially, and the results appear to the user as a single action. This can be used to open a document to a certain page and zoom level, for example.
- Page numbers are *zero-based*: the first page in a document is page 0.
- Quotation marks are needed only if a parameter contains white space.
- The document manipulation methods (deleting pages or scrolling, for example), only work on documents that are already open.

5.3 Acrobat Viewer DDE Messages

This section lists all DDE messages. For complete descriptions of the parameters associated with DDE messages, see the DDE sections of Technical Note #5165, [Acrobat Interapplication Communication Reference](#).

5.3.1 Application Configuration

- *AppExit* — Exits the Acrobat viewer.
- *AppHide* — Iconifies or hides the Acrobat viewer.
- *AppShow* — Shows the Acrobat viewer.
- *CloseAllDocs* — Closes all open documents.
- *FullMenus* — Displays full menus and sets this option in the Acrobat viewer's preferences file. With Acrobat 3.0 or later, all menus are displayed, so this function does nothing.
- *HideToolbar* — Hides the toolbar.
- *MenuItemExecute* — Invokes a menu item, given its language-independent name.
- *ShortMenus* — Displays short menus and sets this option in the Acrobat viewer's preferences file. With Acrobat 3.0 or later, all menus are displayed, so this function does nothing.
- *ShowToolbar* — Shows the toolbar.

5.3.2 Document Manipulation

- *DocClose* — Closes the file without saving it and without prompting the user to save the document if it has been modified.
- *DocDeletePages* — Deletes a specified range of pages in a document. It cannot delete all pages in a document.
- *DocInsertPages* — Inserts specified pages from one file into another.
- *DocOpen* — Opens a document and adds it to the list of documents known to DDE, allowing it to be manipulated by other DDE messages (cf *FileOpen()*).
- *DocReplacePages* — Replaces specified pages using pages from another file.
- *DocSave* — Saves the specified file.

- *DocSaveAs* — Saves an open file into a new file, without warning the user if there is a problem saving.
- *DocSetViewMode* — Controls whether bookmarks, thumbnail images, or neither are shown in addition to the document.
- *FileOpen* — Opens and displays a file, making it the current document and bringing it to the front if it is already open.
- *FileOpenEx* — Opens and displays a file, making it the current document and bringing it to the front if it is already open. File is opened during an idle loop to allow DDE messages to continue flowing during the opening of large documents.

5.3.3 Document Printing

- *DocPrint* — Prints a specified range of pages from a document, without displaying any modal Print dialog to the user.
- *FilePrint* — Prints all pages in a document, displaying a modal Print dialog to the user.
- *FilePrintEx* — Prints all pages in a document, displaying a modal Print dialog to the user. Only PostScript Level 1 operators are used for PostScript printing. Printing is performed during an idle loop to allow DDE messages to continue flowing during the printing of large documents.
- *FilePrintSilent* — Prints all pages in a document, displaying *no* print dialog to the user.
- *FilePrintSilentEx* — Prints all pages in a document, displaying *no* print dialog to the user. Only PostScript Level 1 operators are used for PostScript printing. Printing is performed during an idle loop to allow DDE messages to continue flowing during the printing of large documents.
- *FilePrintTo* — Prints all pages in a document to a specified printer, using a specified driver and port, displaying a modal Print dialog to the user.

- *FilePrintToEx* — Prints all pages in a document to a specified printer, using a specified driver and port., displaying a modal Print dialog to the user. Only PostScript Level 1 operators are used for PostScript printing. Printing is performed during an idle loop to allow DDE messages to continue flowing during the printing of large documents.

5.3.4 View Manipulation

- *DocGoTo* — Goes to the specified page.
- *DocGoToNameDest* — Goes to the specified name destination within the document.
- *DocPageDown* — Scrolls forward through the document by “one screenfull;” same as the “Page Down” keyboard accelerator
- *DocPageLeft* — Scrolls to the left by a small amount; same as “Shift left arrow” keyboard accelerator.
- *DocPageRight* — Scrolls to the right by a small amount; same as “Shift right arrow” keyboard accelerator.
- *DocPageUp* — Scrolls backward through the document by “one screenfull;” same as the “Page Up” keyboard accelerator.
- *DocScrollTo* — Scrolls the view of the current page to a specified location.
- *DocZoomTo* — Sets the zoom for a specified document.

5.3.5 Search-related

- *DocFind* — Finds a string in a specified file. This does not use the cross-document search facility in version 2.0 of Acrobat, but performs a page-by-page search of the specified file.

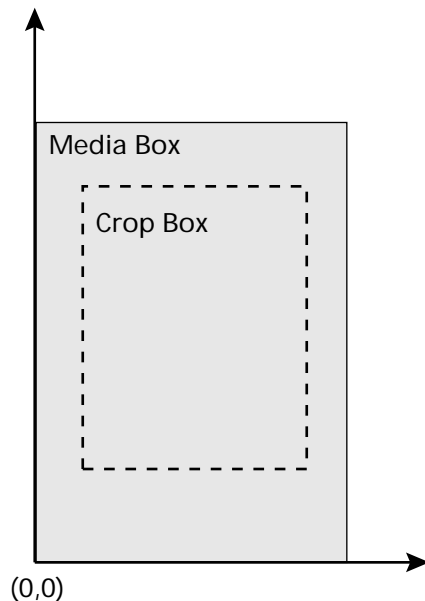
Coordinate Systems

The Acrobat viewer's IAC support uses two coordinate systems: user space and device space. This appendix describes these coordinate systems.

A.1 User Space

User space is the coordinate system used within PDF files. In the IAC interface, it is used for most PDMModel objects (that is, objects such as PDBookmark whose names begin with "PD"). [Figure 4, "User Space Coordinate System"](#) shows the user space coordinate system. The orientation, origin, and scale of the user space coordinate system can be changed by operators in the page description in a PDF file.

Figure 4 User Space Coordinate System

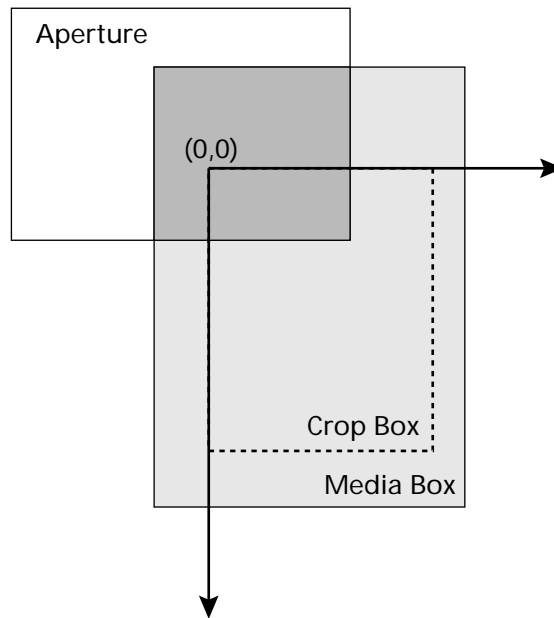


Default user space is the user space coordinate system in effect immediately before each page begins drawing. The origin of this coordinate system is the lower left corner of a page's media box. The x-coordinate increases to the right, and the y-coordinate increases upward. One unit in default user space is 1/72 of an inch.

A.2 Device Space

Device space specifies coordinates in screen pixels, as shown in Figure 5, "Device Space Coordinate System". It is used in the AVModel portion of the IAC interface (that is, objects such as *AVDoc* whose names begin with "AV").

Figure 5 Device Space Coordinate System



The origin of the device space coordinate system is at the upper left corner of the visible page on the screen (that is, the upper left corner of the white part of the page). The x-coordinate increases to the right, and the y-coordinate increases downward.

Note: The upper left corner of the visible page is determined by the intersection of a page's PDF crop box and media box. As a result, the device space coordinate system changes if the cropping on a page changes.

APPENDIX B

Changes Since Earlier Versions

Changes since 13 February 1998 version

Added Visual Basic information. Added descriptions of new OLE automation methods: *Minimize*, *Restore*, *CropPage*, and *CropPages*. Added descriptions of new DDE messages: *DocGoToNameDest*, *FileOpenEx*, *FilePrintEx*, *FilePrintSilentEx*, and *FilePrintToEx*. Added to the list of Acrobat Reader supported DDE messages: *CloseAllDocs*, *DocClose*, *DocGoTo*, *DocGoToNameDest*, *DocOpen*, and *FilePrintEx*.

Changes since 3 October 1997 version

Added *PDDoc.SetOpenInfo*.

Changes since 12 August 1997 version

Clarified how to get an LPDISPATCH for objects.

Changes since 3 June 1996 version

Added information on new DDE methods.

Changes since 14 April 1996 version

Acrobat Exchange 3.0J and 3.01 additions.

Changes since 14 February 1996 version

Minor corrections.

Changes since 13 November 1996 version

Minor corrections.

Changes since 18 July 1996 version

Added changes for Acrobat 3.0. See Technical Note #5165, [*Acrobat Interapplication Communication Reference*](#).

Changes since 24 May 1996 version

Clarified that OLE automation message scheme is synchronous to prevent applications from sending a message before the previous one was completely handled.

Changes since 9 April 1996 version

Added new OLE methods for Acrobat 3.0: *DrawEx*, *OpenInWindowEx*, and *GetAperture*. Made a few updates to DDE section, such as 32-bit support.

Changes since 13 February 1996 version

Updated to clarify page numbering.

Changes since 31 January 1996 version

Updated to reflect UNIX Acrobat version 2.1.

Changes since 15 September 1995 version

- Added chapter on objects.
- Added material to OLE chapter.
- Removed references to Exchange LE 2.0.

Changes since 10 April 1995 version

Obsolete Technical Note #5155, *Acrobat Viewer Interapplication Communication Support*, was split into this technical note, [Acrobat Interapplication Communication Overview](#), and Technical Note #5165, [Acrobat Interapplication Communication Reference](#).

Changes since 07 December 1994 version

- Added description of SimpleQuery item in Search plug-in's DDE support.
- Added appendix describing user and device space coordinate systems.
- Various minor rewording and correction of typos.
- Specify which menu items are present in Exchange LE and which are not.