# Contents

## Preface
- Description ......................................................... 11
- Audience ............................................................ 11
- Resources and Samples ............................................. 11
- How This Document Is Organized ................................ 12
- Conventions Used in This Book ................................. 14

## Chapter 1 Getting Started ........................................... 15
- What Is Acrobat? .................................................... 15
- What Is the Acrobat SDK? ........................................ 17
- What’s New in PDF 1.6? ............................................ 18

## Chapter 2 Technologies Supported by the Acrobat SDK ............. 19
- Acrobat JavaScript .................................................. 19
  - Acrobat JavaScript Objects ...................................... 19
  - Acrobat JavaScript Contexts .................................... 20
- Plug-ins ............................................................... 20
  - Certified Plug-ins ................................................. 21
  - Adobe Reader-enabled Plug-ins ................................ 21
- Development Environments ....................................... 22
- The Acrobat Core API .............................................. 22
- Extended APIs For Plug-ins ...................................... 24
- Interapplication Communication (IAC) .......................... 25
  - Development Environments ..................................... 25
  - Interapplication Communication Objects ..................... 26
  - Using the Acrobat Distiller API ............................... 26
  - IAC and Windows ............................................... 26
  - Apple Event Support ............................................ 27
  - Using Plug-ins for Interapplication Communication .......... 27
  - Technical Limitations of Using the IAC API With Adobe Reader 28
- The Adobe PDF Library ............................................. 28
  - The Adobe PDF Library and the Adobe Acrobat SDK .......... 28
# Acrobat SDK User's Guide

## Chapter 3 Deciding Which Acrobat SDK Technology to Use

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started</td>
<td>29</td>
</tr>
<tr>
<td>When to Use Interapplication Communication</td>
<td>30</td>
</tr>
<tr>
<td>Viewing PDF Documents from an External Application</td>
<td>30</td>
</tr>
<tr>
<td>Controlling Acrobat From an External Application</td>
<td>30</td>
</tr>
<tr>
<td>Deciding Between Plug-ins and JavaScript</td>
<td>31</td>
</tr>
<tr>
<td>General Issues</td>
<td>32</td>
</tr>
<tr>
<td>What You Can Do with Plug-ins That You Can’t Do with JavaScript</td>
<td>33</td>
</tr>
<tr>
<td>What You Can Do with JavaScript That You Can’t Do with a Plug-in</td>
<td>33</td>
</tr>
<tr>
<td>Samples Provided with the Acrobat SDK</td>
<td>34</td>
</tr>
<tr>
<td>Plug-in Samples</td>
<td>34</td>
</tr>
<tr>
<td>Acrobat JavaScript Samples</td>
<td>35</td>
</tr>
<tr>
<td>Interapplication Communication Samples</td>
<td>35</td>
</tr>
</tbody>
</table>

## Chapter 4 Creating PDF Documents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating PDF Documents with Acrobat</td>
<td>37</td>
</tr>
<tr>
<td>Acrobat PDFMaker</td>
<td>37</td>
</tr>
<tr>
<td>Adobe PDF Printer</td>
<td>38</td>
</tr>
<tr>
<td>Save As or Export</td>
<td>38</td>
</tr>
<tr>
<td>Creating PDF Documents from an Authoring Application</td>
<td>39</td>
</tr>
<tr>
<td>Acrobat Distiller</td>
<td>39</td>
</tr>
<tr>
<td>Apple Event Support</td>
<td>40</td>
</tr>
<tr>
<td>Distiller Automation on Windows Using OLE Automation</td>
<td>40</td>
</tr>
<tr>
<td>Automated, Silent Creation of PDF Documents from Windows</td>
<td>41</td>
</tr>
<tr>
<td>Choosing an Adobe PDF Settings File</td>
<td>41</td>
</tr>
<tr>
<td>Automatically Generating Advanced Acrobat Features</td>
<td>41</td>
</tr>
<tr>
<td>Inserting PostScript Code into a Job</td>
<td>42</td>
</tr>
<tr>
<td>Attaching the Native Document to the PDF File</td>
<td>42</td>
</tr>
<tr>
<td>Batch Processing with Distiller</td>
<td>42</td>
</tr>
<tr>
<td>Creating Tagged PDF Documents</td>
<td>43</td>
</tr>
<tr>
<td>PDF Logical Structure</td>
<td>43</td>
</tr>
<tr>
<td>The Structure Tree</td>
<td>43</td>
</tr>
<tr>
<td>Adding Structure Data to a PDF Document</td>
<td>44</td>
</tr>
<tr>
<td>Using pdfmark to Add Structure Data to PDF</td>
<td>44</td>
</tr>
<tr>
<td>Creating PDF Documents Using Plug-ins or Acrobat JavaScript</td>
<td>45</td>
</tr>
<tr>
<td>Creating an Empty PDF File</td>
<td>45</td>
</tr>
<tr>
<td>Creating PDF Files from Multiple Files</td>
<td>45</td>
</tr>
<tr>
<td>Creating PDF Files Without Using Acrobat</td>
<td>46</td>
</tr>
</tbody>
</table>
Chapter 5  Working with PDF Documents  ................................................. 47

Automating Processes: Opening PDF Documents Silently .................. 48
Adding Navigation to PDF Documents .............................................. 49
Adding and Managing Bookmarks .................................................. 49
Adding and Managing Thumbnails .................................................. 50
Adding and Managing Links .......................................................... 50
Using Actions for Special Effects .................................................... 51
Manipulating Pages of a PDF File ..................................................... 51
Creating and Editing Page Content ................................................... 52
PDFEdit API ..................................................................................... 52
PDFEdit Samples ............................................................................. 52
Using Document Logical Structure .................................................. 53
Adding Structure Data to a PDF Document ........................................ 53
Using pdffmark to Add Structure Data to PDF ................................. 53
Modifying PDF Documents in Other Ways ........................................ 54
Adding Watermarks ......................................................................... 54
Fixed Printing for Engineering Documents ....................................... 54
Spell-Checking .................................................................................. 55
Using Multimedia ............................................................................. 55
Interacting with OBDC Databases .................................................... 56
Printing PDF Files ............................................................................ 56
Customizing and Automating Printing ............................................... 56
Working with Embedded Fonts ........................................................ 57
Large Page Sizes in PDF Documents ................................................. 58
Wireframe Drawing .......................................................................... 58

Chapter 6  Modifying the User Interface ............................................. 59

Adding or Removing Menus and Menu Items .................................... 59
Adding Menu Items .......................................................................... 59
Adding a New Menu ......................................................................... 60
Modifying the Toolbar ....................................................................... 60
Adding Items to an Existing Toolbar .................................................. 60
Creating a New Toolbar ..................................................................... 60
Using the Adobe Dialog Manager (ADM) .......................................... 61
ADM Controls .................................................................................. 61
### Contents

ADM Objects ................................................................. 62
Using ADM from Acrobat JavaScript .......................... 63
Using the ADM API from a Plug-in ......................... 63
Customizing Acrobat Help ........................................... 64
Controlling the “About” Box and Splash Screen .......... 64
Placing Plug-in Help Files in a Standard Location .... 64
Customizing the How-To Panel ................................. 65

**Chapter 7** Using Online Collaboration and Annotations ........ 67
Using Online Collaboration ........................................... 67
How FDF Files Are Used ............................................ 68
Online Collaboration and Acrobat JavaScript ............ 68
Extensibility ............................................................... 68
Using Annotations in Document Workflows ................ 69
Annotations and JavaScript ......................................... 69
Annotations and Plug-ins ............................................ 70
Annotations and the IAC API ......................................... 70
Creating New Annotation Types ..................................... 70
Using Stamps in Approval Workflows ............................ 71

**Chapter 8** XML and the Acrobat SDK ......................... 73
The Adobe XML Architecture ......................................... 73
XML Forms ................................................................. 74
XML Templates ........................................................... 74
Extensible Metadata Platform (XMP) ......................... 74
SOAP and Web Services ................................................ 74
Converting PDF Documents to XML Format ................ 76
Converting to XML Format Using Acrobat JavaScript .... 76
The SaveAsXML Plug-in ............................................... 76
Managing XML-based Information ............................... 77

**Chapter 9** Forms and the Acrobat SDK ....................... 79
Types of Forms ............................................................ 79
XML Forms ................................................................. 79
Acrobat Forms ............................................................ 80
Forms Workflows ......................................................... 81
Elements of Acrobat Forms ......................................... 81
Guidelines for Creating a New Form ..................................................... 82
Acrobat Forms and the Acrobat SDK .................................................. 82
  Acrobat JavaScript for Acrobat Forms ......................................... 82
  The Forms API ........................................................................... 83
  OLE Automation for Acrobat Forms ........................................... 83
XML Forms and the Acrobat SDK ...................................................... 84
Automatically Populating Forms ...................................................... 84
Creating Web-Ready Forms ............................................................. 84
Punchcard/Barcode Form Fields ....................................................... 84
Collecting Data from Completed Forms ......................................... 85
  XML and XDP Formats ............................................................... 85
  FDF and XFDF Formats ............................................................. 85
Saving Data to a Microsoft Excel Spreadsheet .................................. 85
Submitting Data from Attachments .................................................. 85
Exporting Data as Images ............................................................... 85
XML Form Templates ...................................................................... 86

Chapter 10  Rights-enabled PDF Documents ................................. 87
  Additional Usage Rights ............................................................... 87
  Differences between Usage Rights and Reader Enabling ................ 88
  Creating Rights-enabled Documents .......................................... 89
  Accessing Additional Usage Rights ............................................. 89
  Applying and Validating Additional Usage Rights ......................... 90
  Effects of Reader Extensions on Acrobat Standard and Acrobat Professional ...................................................... 90
  Enabling Collaboration ................................................................... 90

Chapter 11  Providing Document Security ..................................... 91
  Acrobat Security Overview .......................................................... 91
  Encryption and Decryption .......................................................... 92
  Digital IDs and Certificates ......................................................... 92
    Obtaining a Digital ID from a Third Party ................................ 93
    Digital IDs and the Acrobat SDK .............................................. 93
  Security Handlers ................................................................. 94
  PPKLite and PPKMS Security Handlers .................................... 94
  Security Handlers and Plug-ins ................................................... 94
  Security Handlers and Acrobat JavaScript ................................ 95
  Digitally Signing Adobe PDF Documents ..................................... 96
<table>
<thead>
<tr>
<th>Chapter 12</th>
<th>Working with Metadata</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types of Metadata</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>Document Information Dictionary</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>Metadata Streams</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>Extensible Metadata Platform (XMP)</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>Accessing XMP Metadata Using Acrobat JavaScript</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>Accessing XMP Metadata From a Plug-in</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>Acrobat Distiller and XMP Metadata</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>The Adobe XMP Toolkit</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>Searching Metadata</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>Object Data</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>Creating Object Data</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>Searching and Accessing Object Data</td>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 13</th>
<th>Searching and Indexing</th>
<th>107</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Acrobat Search Plug-in</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>Creating Custom Searches</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>Search Options</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>Searching across Multiple Documents.</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>Replacing the Acrobat Search Plug-in</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>Indexing PDF Documents</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>Extracting and Highlighting Text</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>Indexing and Acrobat JavaScript</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>The Acrobat Catalog Plug-in</td>
<td>109</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 14</th>
<th>Accessibility</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessibility Overview</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>Optimizing the Accessibility of PDF Documents</td>
<td>112</td>
<td></td>
</tr>
</tbody>
</table>
Making Forms Accessible ......................................................... 112
Field Descriptions ................................................................. 112
Setting Tab Order ................................................................. 113
Annotations and Accessibility ............................................. 113
Tagged Annotations ............................................................... 113
Text-To-Speech .................................................................... 113
Accessibility and Security .................................................... 113
Microsoft’s Active Accessibility (MSAA) Interface ................ 114

Chapter 15 Working With PDF Layers ................................. 115
About Adobe PDF layers ....................................................... 115
Creating Layered PDF Files ................................................. 116
Navigating with Layers ......................................................... 116
Working with Layers from Acrobat JavaScript ..................... 116
Working with Layers from a Plug-in ..................................... 117

Index ................................................................................... 119
Preface

Description

This guide provides an introduction to the Adobe® Acrobat® Software Development Kit (SDK). It provides a general overview of the types of things you can do with the SDK and the technologies that are available to you through the SDK.

Audience

This guide is meant for both experienced Acrobat SDK developers and those who are new to the Acrobat SDK.

For those new to the Acrobat SDK, this guide provides information about Acrobat SDK technologies and the many different ways that developers can extend Acrobat using the SDK.

For developers who have used earlier versions of the SDK, this guide provides information on new and changed features in this version of the Acrobat SDK.

Resources and Samples

The following resources and samples provide further information about the Acrobat SDK:

- Acrobat and PDF Library API Overview provides a high-level overview of the Acrobat and PDF Library APIs.
- Acrobat and PDF Library API Reference contains the method prototypes and details on arguments. By using this reference online, you can copy prototypes directly into your plug-in as you are coding it.
- Acrobat JavaScript Scripting Guide provides an overview of how to use Acrobat JavaScript to develop and enhance standard workflows.
- Acrobat JavaScript Scripting Reference describes in detail all objects, properties and methods within the Acrobat extension to JavaScript, and gives code examples.
- Acrobat Plug-in Guide explains how to use the Acrobat core and extended APIs to write plug-ins for Acrobat and Adobe Reader. It describes basic Acrobat development concepts and provides information on upgrading plug-ins from Acrobat 6 to Acrobat 7.
- Acrobat Interapplication Communication Overview explains the interapplication communication (IAC) support concepts, such as objects and commands universally understood by applications.
Preface

How This Document Is Organized

- Acrobat Interapplication Communication Reference provides a detailed reference of all the calls needed for Apple® Events, OLE and DDE.
- Guide to SDK Samples provides an overview of the samples provided with the Acrobat SDK.
- SnippetRunner Cookbook provides an overview of the plug-in snippets provided with the Acrobat SDK.
- PDF Reference, fifth edition, Version 1.6 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, or who wish to gain a detailed understanding of the PDF file format.
- Adobe Dialog Manager Programmer’s Guide and Reference describes how to create platform-independent dialog boxes for your plug-in.
- Developing for Adobe Reader provides an introduction to those portions of the Adobe Acrobat Software Development Kit (SDK) that pertain to your development efforts for Adobe Reader.

Additional documents that you should have available for reference are listed below.
- Adobe Reader-enabled Plug-ins describes the steps required to enable a plug-in to be loaded by Adobe Reader.
- PostScript Language Reference, third edition describes the syntax and semantics of the PostScript language and the Adobe imaging model.
- The samples and snippets included in the Acrobat SDK. For more information, visit the Adobe Solutions Network at http://partners.adobe.com/asn/.

How This Document Is Organized

This document is organized as follows:
- Chapter 1, “Getting Started,” introduces the Acrobat SDK and summarizes new features in the Acrobat SDK and in PDF 1.6.
- Chapter 2, “Technologies Supported by the Acrobat SDK,” describes the technologies that make up the Acrobat SDK.
- Chapter 3, “Deciding Which Acrobat SDK Technology to Use,” describes general issues that factor into decisions about which Acrobat SDK technology to use.
- Chapter 4, “Creating PDF Documents,” provides information about how to use the Acrobat SDK to create PDF documents.
- Chapter 5, “Working with PDF Documents,” describes how to use the Acrobat SDK to add navigation, how to modify and manipulate PDF documents, and how to print PDF documents.
- Chapter 6, “Modifying the User Interface,” describes how to change the Acrobat or Adobe Reader user interface using the Acrobat SDK.
- Chapter 7, “Using Online Collaboration and Annotations,” describes online collaboration workflows, annotations, and stamps.
- Chapter 8, “XML and the Acrobat SDK,” describes the Adobe XML architecture and the role that Acrobat and the Acrobat SDK play in XML-based workflows.
- Chapter 9, “Forms and the Acrobat SDK,” describes how to extend the functionality of forms using both Acrobat and XML Forms.
- Chapter 10, “Rights-enabled PDF Documents,” describes how to work with PDF documents with additional usage rights in Acrobat and Adobe Reader.
- Chapter 11, “Providing Document Security,” describes how to use the Acrobat SDK to provide security to PDF documents.
- Chapter 12, “Working with Metadata,” describes how to access and manipulate document metadata and object data using the Acrobat SDK.
- Chapter 13, “Searching and Indexing,” describes how to customize Acrobat’s searching and indexing capabilities using the Acrobat SDK.
- Chapter 14, “Accessibility,” describes Acrobat and Acrobat SDK support for accessibility of documents to visually-impaired users.
## Conventions Used in This Book

The Acrobat documentation uses text styles according to the following conventions:

<table>
<thead>
<tr>
<th>Font</th>
<th>Used for</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>monospaced</td>
<td>Paths and filenames</td>
<td>C:\templates\mytmpl.fm</td>
</tr>
<tr>
<td></td>
<td>Code examples set off from plain text</td>
<td>These are variable declarations: AVMenu commandMenu, helpMenu;</td>
</tr>
<tr>
<td>monospaced bold</td>
<td>Code items within plain text</td>
<td>The GetExtensionID method ...</td>
</tr>
<tr>
<td></td>
<td>Parameter names and literal values in reference documents</td>
<td>The enumeration terminates if proc returns false.</td>
</tr>
<tr>
<td>monospaced italic</td>
<td>Pseudocode</td>
<td>ACCB1 void ACCB2 ExeProc(void) { do something }</td>
</tr>
<tr>
<td></td>
<td>Placeholders in code examples</td>
<td>AFSimple_Calculate(cFunction, cFields)</td>
</tr>
<tr>
<td>blue</td>
<td>Live links to Web pages</td>
<td>The Adobe Solutions Network URL is: <a href="http://partners.adobe.com/">http://partners.adobe.com/</a></td>
</tr>
<tr>
<td></td>
<td>Live links to sections within this document</td>
<td>See Using the SDK.</td>
</tr>
<tr>
<td></td>
<td>Live links to code items within this document</td>
<td>Test whether an ASAtom exists.</td>
</tr>
<tr>
<td>bold</td>
<td>PostScript language and PDF operators, keywords, dictionary key names</td>
<td>The setpagedevice operator</td>
</tr>
<tr>
<td></td>
<td>User interface names</td>
<td>The File menu</td>
</tr>
<tr>
<td>italic</td>
<td>Document titles that are not live links</td>
<td>Acrobat and PDF Library API Overview</td>
</tr>
<tr>
<td></td>
<td>New terms</td>
<td>User space specifies coordinates for...</td>
</tr>
<tr>
<td></td>
<td>PostScript variables</td>
<td>filename deletefile</td>
</tr>
</tbody>
</table>
Getting Started

This chapter introduces you to Acrobat and the Acrobat SDK, and summarizes the changes to the SDK in the current release.

The chapter consists of the following sections:

- What Is Acrobat?
- What Is the Acrobat SDK?
- What's New in PDF 1.6?

What Is Acrobat?

Adobe ® Acrobat ® 7.0 consists of a family of products for creating, modifying, indexing, searching, displaying, and manipulating PDF (Portable Document Format) files. For further information about the Acrobat family of products, see http://partners.adobe.com/links/acrobat.

What is the Acrobat Family?

- Adobe Reader for viewing, navigating, and printing PDF documents. Adobe Reader is free software that lets anyone view and print Adobe PDF files on all major computer platforms, as well as fill in and submit Adobe PDF forms. Adobe has distributed more than 500 million copies of the software worldwide.

- Acrobat for adding navigational links, annotations, and security options, in addition to the functionality provided by Adobe Reader. Acrobat 7.0 is offered in two configurations:
  - Acrobat 7.0 Standard — Offers all of the Adobe PDF creation and distribution tools in Acrobat Elements ® (see below) and also allows users to more effectively manage document review cycles. Acrobat Standard's intuitive review and commenting tools help streamline collaboration and reduce the costs and inefficiencies of paper-based review processes.
  - Acrobat 7.0 Professional — Contains all of the functionality of Acrobat 7.0 Standard and Acrobat Elements (see below). In addition, it offers the most advanced control over document creation, exchange, review, and output. It is intended for users who need to perform sophisticated tasks, such as preflighting print files, distributing layered CAD drawings, and creating interactive forms.
Getting Started

What Is Acrobat?

- Acrobat Elements for organizations requiring widespread PDF creation capability without the need to install the full Acrobat/Distiller product on every desktop. Acrobat Elements allows organizations to extend their investments in Microsoft® Office® by standardizing on Adobe PDF for document distribution. Acrobat Elements is available through licensing with minimum orders of 1,000 seats. There is no public API for Acrobat Elements.

What Are the Tools For Creating PDF Files?

- The Adobe Acrobat family. Adobe Acrobat 7.0 Standard and Professional both include Acrobat Distiller® for creating PDF files from PostScript files created with your favorite applications (including desktop publishing software).

- Acrobat Capture® with OCR (Optical Character Recognition) for creating text-searchable PDF documents from scanned paper originals.

- The Adobe LiveCycle™ products, which provide core document services designed to align with your organization’s services-oriented architecture and leverage open standards including J2EE and XML. They provide key functions such as document generation, document control and security, document collaboration, and process management for electronic forms. These products include Adobe LiveCycle Forms® and Adobe Document Server®.

Related Adobe Technologies

- Adobe LiveCycle™ Reader Extensions® provides advanced services and functionality for documents and forms accessed by Adobe Reader clients, extending users’ capabilities for participating in form applications or workflows without requiring the full Adobe Acrobat product.

- Adobe Policy Server® enables document authors and IT administrators to create usage policies that dynamically control who can view a PDF document, determine whether the recipient can modify, copy, print, or forward the document, and establish document expiration dates.

What Is the Acrobat SDK?

The Acrobat SDK is a set of tools that help you develop software that interacts with Acrobat technology. The SDK contains header files, type libraries, simple utilities, sample code, and documentation.

Using the Acrobat SDK, you can develop software that integrates with Acrobat and Adobe Reader in the following ways:

<table>
<thead>
<tr>
<th>Acrobat JavaScript</th>
<th>Write Acrobat JavaScript scripts, either in an individual PDF document or externally, to extend the functionality of Acrobat or Adobe Reader.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug-ins</td>
<td>Create plug-ins that are dynamically linked to and extend the functionality of Acrobat or Adobe Reader.</td>
</tr>
<tr>
<td>Interapplication Communication (IAC)</td>
<td>Write a separate application process that uses interapplication communication (IAC) to control Acrobat functionality. DDE and OLE are supported on Windows, and Apple events/AppleScript on the Macintosh. IAC is not available on Unix®.</td>
</tr>
</tbody>
</table>
What’s New in PDF 1.6?

The following features are new in the PDF 1.6 specification. For complete details, see the PDF 1.6 Reference.

- An enhancement to the syntax of DeviceN color spaces to support NChannel color spaces, which give applications greater flexibility when representing colors that are not available on a target device.
- Encryption enhancements to support using the AES encryption algorithm and to selectively encrypt embedded files.
- Enhancements to digital signatures related to usage rights and MDP (modification detection and prevention).
- Additional support for embedded file attachments, including cross-document linking to and from embedded files.
- The ability to define user properties that contain non-drawing-related information about objects on a page.
- The ability to accurately specify relationships between the dimensions of objects on a page and their real-world counterparts.
- An increase to the maximum page size of a PDF document, by specifying the size of the unit in default user space.
- The ability to specify objects that should be printed at a specific size, regardless of the dimensions of the printed page.
- New facilities for embedding OpenType fonts in PDF files.
- Enhancements to markup annotations, including the ability to group them.
- The ability to specify non-rectangular regions for link annotations.
- Greater flexibility in specifying the visibility of graphics based on the state of optional content groups, and the ability to lock the states of groups.
- The addition of Tagged PDF attributes relating to the layout of content in columns.
- The ability to incorporate models of three-dimensional graphical data, using the 3DIF format, in a PDF file.
Technologies Supported by the Acrobat SDK

As mentioned in “What Is the Acrobat SDK?” on page 17, you can develop software that integrates with Acrobat and Adobe Reader in three ways:

- Acrobat JavaScript
- Plug-ins
- Interapplication Communication (IAC)

The Acrobat SDK provides support for development on both Windows® and Apple Macintosh® environments. A Unix version of the Acrobat SDK is also available from http://partners.adobe.com/asn. It has release notes containing UNIX-specific information.

Besides the technologies provided by the Acrobat SDK, you can also use the Adobe PDF Library (PDFL) to develop applications that create and manipulate PDF documents but do not interact with Acrobat. The Adobe PDF Library is licensed separately from the Acrobat SDK. An overview of the PDF Library is provided in “The Adobe PDF Library” on page 28.

**Acrobat JavaScript**

Acrobat JavaScript is a cross-platform scripting language. Through its JavaScript extensions, Acrobat exposes much of the functionality of Acrobat and its plug-ins to the document author.

Acrobat JavaScript objects, properties and methods can also be accessed through Visual Basic or C# to automate the processing of PDF documents. See “Accessing the JSObject Interface” on page 27.

For complete information on Acrobat JavaScript objects, see the Acrobat JavaScript Scripting Guide and the Acrobat JavaScript Scripting Reference.

**Acrobat JavaScript Objects**

Acrobat JavaScript defines several objects that allow your code to interact with the Acrobat application, a PDF document, or fields within a PDF document. The most commonly used objects control the Acrobat or Adobe Reader application, the Acrobat JavaScript console, the PDF document, the Adobe Dialog Manager, SOAP web services, databases, security, searches, and JavaScript events.

For a complete list of Acrobat JavaScript objects, see the Acrobat JavaScript Scripting Reference.
Acrobat JavaScript Contexts

Acrobat JavaScripts can be applied at a variety of levels:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folder level</td>
<td>These scripts contain variables and functions that may be generally useful to the Acrobat application, and are visible from all documents.</td>
</tr>
<tr>
<td>document level</td>
<td>These scripts are function and variable definitions that are generally useful to a given document, but are not applicable outside the document.</td>
</tr>
<tr>
<td>field level</td>
<td>These scripts can be associated with any component within a document. In the case of dynamic XML forms, such scripts are only visible to that component and execute as actions or events. See “XML Forms” on page 79.</td>
</tr>
<tr>
<td>batch level</td>
<td>These scripts can be applied to a collection of documents, and operate at the application level.</td>
</tr>
</tbody>
</table>

Each of these levels represents a context in which processing occurs, which affects when the scripts are loaded and how they are accessed inside and outside documents.

The placement of a JavaScript at a given level also determines its reusability. Folder level scripts are available within all documents, document level scripts are available to all fields within a given document, and field level scripts are visible only to the fields with which they are associated.

For more information, see the Acrobat JavaScript Scripting Guide.

Plug-ins

Plug-ins are dynamically-linked extensions to Acrobat or Adobe Reader. They can hook into the user interface in a number of ways and can register to be called when a variety of events occur in the application.

A plug-in is a program written in ANSI C/C++ that uses the Acrobat public APIs. It adds functionality to Acrobat Professional, Acrobat Standard, or Adobe Reader. A plug-in program file goes into a Plug_ins folder or directory and is initialized during Acrobat or Adobe Reader startup.
There are three types of plug-ins:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Acrobat plug-ins</td>
<td>These plug-ins run on Acrobat Professional and Acrobat Standard. Plug-ins for Acrobat Professional can use any of the Acrobat SDK APIs. Plug-ins for Acrobat Standard do not have access to some APIs. See the Acrobat Plug-in Guide for more information.</td>
</tr>
<tr>
<td>Adobe Reader-enabled plug-ins</td>
<td>These plug-ins use a restricted set of APIs. Adobe Reader-enabled plug-ins are developed with permission from Adobe and require special processing to load under Adobe Reader. See “Adobe Reader-enabled Plug-ins” on page 21 for more information.</td>
</tr>
<tr>
<td>Certified plug-ins</td>
<td>These plug-ins have undergone extensive testing to ensure that they do not compromise the integrity of Acrobat’s security model. See “Certified Plug-ins” on page 21.</td>
</tr>
</tbody>
</table>

Plug-ins are:
- DLLs on Windows. Note, however, that plug-in names must end in .API, not .DLL.
- Code fragments on Mac OS X.
- Shared libraries on UNIX.

**Certified Plug-ins**

Certified plug-ins have undergone extensive testing by Adobe to ensure that they do not compromise the integrity of Acrobat’s security model. There is a checkbox in the Acrobat and Adobe Reader user interface that can be used to ensure that only certified plug-ins load. This means that other plug-ins will not load. Certified plug-ins can be provided only by Adobe.

**Adobe Reader-enabled Plug-ins**

Adobe Reader only accepts Reader-enabled plug-ins. Adobe Reader-enabled plug-ins can only access a limited set of APIs. Reader-enabled plug-ins require permission and licensing from Adobe. For information on enabling plug-ins for Adobe Reader, see the document, Adobe Reader-enabled Plug-ins.

Plug-ins for Adobe Reader can use additional APIs if the PDF document has additional usage rights. For more information, see Chapter 10, “Rights-enabled PDF Documents”.

Acrobat SDK User’s Guide  

21
Development Environments

Windows

Windows developers can develop plug-ins using C and C++ with Visual Studio .NET 2003. There is currently no support for development of plug-ins using managed languages such as C# or VB.NET.

However, managed languages are fully supported for use with interapplication communication (IAC) and specifically with the JSObject. This enables those languages to take full advantage of Acrobat’s functionality via the JavaScript bridge. For more information, see “Interapplication Communication (IAC)” on page 25.

Mac OS X

Beginning in Acrobat 7.0, all plug-ins developed on the Mac OS X must use the Mach-O runtime architecture. A plug-in can be built as either a bundle or a framework. Apple Xcode 1.2 or 1.5 is required because SDK projects depend on certain header files that are included with the Xcode development environment. However, The Acrobat SDK does not support development with the Xcode tools.

Plug-ins developed in previous SDK versions must be rebuilt under Mach-O. For more information on migrating plug-ins from previous versions of Acrobat, see the Acrobat SDK Plug-in Guide.

The Acrobat Core API

Plug-ins access and control the resources of the Acrobat application host environment using the Acrobat Core API. The core API consists of a set of methods that operate on objects. The objects have types and encapsulate their data. This object orientation is a conceptual model, implemented using a standard ANSI C programming interface. Methods are C functions; objects are opaque data types. The Core API is supported on Microsoft 32-bit Windows, Apple Macintosh, and UNIX platforms.
The core API methods are organized into the hierarchy shown in Figure 2.1.

**Figure 2.1  Overview of Core API**

![Diagram showing the hierarchy of the core API methods]

**Acrobat Viewer Layer**
The Acrobat Viewer (AV) layer, also known as AcroView or AV Model, works with the Acrobat or Adobe Reader application. Its methods allow plug-ins to manipulate components of the Acrobat or Adobe Reader application itself, such as menus and menu items.

**Portable Document Layer**
The Portable Document (PD) layer, also known as PDMModel, provides access to components of PDF documents. Its methods allow plug-ins to manipulate document components such as document pages and annotations. Two method groups are closely related to the PD layer, each controlling a different aspect of a PDF document:

- **PDFEdit** methods deal with the physical representation of a PDF document. More specifically, PDFEdit methods treat page content as a list of objects whose values and attributes can be modified by a plug-in. The methods allow your plug-in to read, write, edit, and create page contents and page resources, which may contain objects such as fonts and images.

- **PDSEdit** methods deal with the logical structure-tree information of a PDF document. A PDF document’s logical structure is built independently of its physical representation, with pointers between the logical structure and the physical representation. PDSEdit methods allow your plug-in to add, modify, navigate, search and extract data from PDF documents. For example, PDSEdit methods can obtain logically-ordered content, independently of drawing order.

For more information about logical structure and tagged PDF files, see “Creating Tagged PDF Documents” on page 43.
Acrobat Support

The Acrobat Support (AS) layer provides platform-independent utility functions and allows plug-ins to override the built-in file-handling mechanisms.

Cos Layer

The Cos Object System (Cos) layer provides access to the building blocks used to construct documents. Cos methods allow plug-ins to manipulate low-level data such as dictionary and string objects in a PDF file.

Whenever possible, you should use higher level APIs to access and manipulate PDF files. Though you can use the Cos layer APIs to perform most types of access or manipulation of a PDF file, it can be very difficult and requires in-depth knowledge of PDF file structure.

Platform-Specific Methods

In addition to the method groups represented in Figure 2.1, the core API includes platform-specific plug-in utilities to handle issues that are unique to Macintosh, Windows and UNIX platforms. For more information, see the Acrobat SDK Plug-in Guide.

Extended APIs For Plug-ins

Plug-ins may expose their own functionality and make it available to other plug-ins, in the same way that Acrobat functionality is available through the core API. For example, Acrobat contains several plug-ins, such as Search and Digital Signatures, whose APIs are available to plug-ins. In fact, the Acrobat architecture encourages the use of plug-ins to “expose” APIs for use by other plug-ins. Acrobat uses many plug-ins to implement features; some of these plug-ins expose their APIs.

API “exposure” is accomplished through a mechanism called the Host Function Table (HFT). A plug-in can export an HFT for use by other plug-ins, and it can import the HFTs of other plug-ins.

To learn how to import and export HFTs, see the Acrobat SDK Plug-in Guide and the Acrobat and PDF Library API Overview.

Plug-in APIs

The following are the plug-ins that export HFTs. For more information, see the Acrobat and PDF Library Overview and the Acrobat and PDF Library Reference.

- The Catalog Plug-in creates full-text indices, searchable with the Acrobat Search plug-in.
- The Digital Signature Plug-in handles the generic functions common to all digital signatures. You can write a plug-in that handles your specific kinds of signatures (signing methods), such as public-private key (PPK), handwriting, retinal scans, or fingerprints.
- The Forms Plug-in helps make Acrobat a more powerful and comprehensive tool for creating and filling out electronic forms. With it, you can convert a PDF file into an electronic form that looks just like the original paper form.
- The PDF Consultant Plug-in allows you to inspect, analyze, and repair PDF documents.
Technologies Supported by the Acrobat SDK

Interapplication Communication (IAC)

- The **Search Plug-in** allows you to make queries, using the index created by Catalog. Using events and DDE messages, remote clients can submit queries and manipulate a list of indexes.
- The **Spelling Plug-in** allows users to spell-check text in comments and form fields.
- The **Weblink Plug-in** adds the ability to link to the World Wide Web. The Weblink driver performs low-level platform-specific functions with a Web browser on behalf of Weblink.
- The **SaveAsXML Plug-in** allows you to extend the “Save as type” choices in the SaveAs dialog box to allow a tagged PDF document to be saved in XML, HTML, or similar text-based formats. See *Using the Save as XML Plug-in*.

Adobe Dialog Manager

The Adobe Dialog Manager (ADM) is a cross-platform API for implementing dialog interfaces for Adobe applications. In Acrobat 5.0 and above, ADM allows plug-in developers to use a single code base to implement dialog behavior on both Macintosh and Windows.

The *Adobe Dialog Manager Programmer’s Guide and Reference* explains the fundamentals of ADM and describes its functions in detail, as well as the specific steps Acrobat plug-in developers must take in order to access ADM functionality.

AcroColor API

The Acrocolor API is an HFT that allows you to access the AcroColor engine (ACE), which controls color profile management. Strictly speaking, it is not an “extended API” since methods are accessed in a manner similar to the core APIs. However, it constitutes its own “layer” in a manner similar to ADM, so it is described in these terms.

Interapplication Communication (IAC)

Acrobat provides support for interapplication communication (IAC) through Apple events and AppleScript on Macintosh platforms, and through OLE automation and DDE on Windows platforms. Adobe Reader also supports IAC, but does not support OLE on Windows.

IAC support allows programs to control Acrobat or Adobe Reader in much the same way a user would. You can also use IAC support to render a PDF file into an external application window instead of the Acrobat window. The IAC support methods and events serve as wrappers for some of the core API calls in the SDK.

Development Environments

On Windows, you can develop IAC applications using any of the following languages:

- Visual Basic .NET
- Visual C++ .NET
- Visual C# .NET
On a Macintosh, you develop IAC applications using CodeWarrior 9.2.

**Interapplication Communication Objects**

You can think of the Acrobat API as having two distinct layers that use IAC objects:
- The Acrobat Viewer (AV) level
- The Portable Document (PD) layer.

An example of an AV layer object is the view of a document. You would expect to see the view of a document object residing in the layer associated with the Viewer. An example of a PD layer is a page in the document.

For more information about the Acrobat PDF layers, see “The Acrobat Core API” on page 22.

**Using the Acrobat Distiller API**

You can use the automation interface of the Acrobat Distiller API to remotely control PDF creation from an external application. For more information, see “Creating PDF Documents from an Authoring Application” on page 39.

**IAC and Windows**

**OLE Support on Windows**

Acrobat is an OLE server and also responds to a variety of OLE automation messages. You can embed PDF documents into documents created by an application that is an OLE client. Adobe Reader does not support OLE on Windows.

For complete descriptions of the Acrobat parameters associated with OLE automation methods, see the OLE automation sections of the *Acrobat Interapplication Communication Reference*.

**PDF Browser Controls**

On Windows, you may use the AcroPDF library to display a PDF document in applications using simplified browser controls. In this case, the PDF is treated as an ActiveX document, and the interface is available in Adobe Reader.

Once the PDF document is loaded, you can implement browser controls to:
- Determine which page to display as well as the display
- Control view and zoom modes
- Determine whether to display bookmarks, thumbs, scrollbars, and toolbars
- Print pages using various options
- Highlight a text selection.

For more information, see the *Acrobat Interapplication Communication Overview*. 
Technologies Supported by the Acrobat SDK

Interapplication Communication (IAC)

Accessing the JSObject Interface

Acrobat 7.0 provides a rich set of JavaScript programming interfaces that are designed to be used from within the Acrobat environment. It also provides a mechanism (known as the JSObject) that allows external clients to access the same functionality from environments such as VB .NET, Visual C++ .NET and Visual C# .NET.

Programming Acrobat JavaScript Using Visual Basic gives you the information you need to get started using the extended functionality of JavaScript from a Visual Basic programming environment. It provides a set of examples to illustrate the key concepts.

In precise terms, the JSObject is an interpretation layer between an OLE Automation client such as a Visual Basic application and the JavaScript functionality provided by Acrobat. From a programmer’s point of view, the end result is that programming JSObject from a Visual Basic environment is quite similar to programming in JavaScript using the Acrobat console.

DDE Support

Dynamic Data Exchange (DDE) is a feature of Windows that allows two programs to share data or send commands directly to each other. The DDE messages that are supported by Acrobat are listed in the Acrobat Interapplication Communication Overview.

IMPORTANT: You should use OLE automation instead of DDE whenever possible, since DDE is not a COM technology.

Apple Event Support

The Acrobat viewers support Apple Events and a number of Apple Event objects on the Macintosh platforms. IAC support includes some of the objects and events described in Apple Event Registry: Standard Suites, as well as Acrobat-specific objects and events. For more information, see the Acrobat Interapplication Communication Overview.

NOTE: You can find information on Apple Events supported by the Acrobat Search plug-in by referring to the Acrobat and PDF Library API Reference. Other plug-ins supporting additional Apple Events are described in the Acrobat and PDF Library API Overview.

When programming for Macintosh platforms, it is advised that you use AppleScript with Acrobat whenever possible. There are some Apple Events not available through AppleScript; these can be handled with C or other programming languages. When programming in C, use the declarations in AcroAETypes.h. For a complete description of the parameters, see the Acrobat Interapplication Communication Reference.

Using Plug-ins for Interapplication Communication

You can extend the functionality of the IAC interfaces by writing plug-ins that use core API objects that are not already part of the IAC support system. The Acrobat SDK provides a sample, DDEServer, which demonstrates this functionality.

See the Acrobat Interapplication Communication Overview for more information.
Technologies Supported by the Acrobat SDK

The Adobe PDF Library

Technical Limitations of Using the IAC API With Adobe Reader

Adobe Reader does not support OLE automation.

Adobe Reader supports only limited DDE message and Apple events. See the Acrobat Interapplication Communication Reference for more details.

There are licensing restrictions on functionality that may be added to Adobe Reader by plug-ins. See http://partners.adobe.com/asn.

The Adobe PDF Library

The Adobe PDF Library is based on the core technology of the Adobe Acrobat line of products and offers complete functionality for generating, manipulating, rendering, and printing Adobe PDF documents.

NOTE: The PDF Library is licensed separately from Acrobat and the Acrobat SDK. For more information, see http://partners.adobe.com/asn.

Designed specifically for OEMs, ISVs, system integrators, and enterprise IT developers, the Adobe PDF Library SDK contains a powerful set of functions for developing third-party solutions and workflows around the Adobe Portable Document Format (PDF).

The library enables Adobe PDF functionality to be seamlessly embedded within applications. It also provides reliable, accurate and Adobe-supported implementation of the latest PDF specification.

The Adobe PDF Library and the Adobe Acrobat SDK

There is a significant overlap between the functionality provided by the PDF Library Software Development Kit (SDK) and by the Acrobat SDK. They differ in providing access to the Acrobat user interface:

● The Acrobat SDK is meant for the plug-in environment, and allows you to control and interact with the Acrobat user interface.

● The PDF Library SDK is intended for interaction between PDF and other applications, such as high volume batch processing and PDF generation applications. It does not export methods for creating or managing Acrobat user interface elements—that is, the AcroView (AV) layer of the core API.

For more information, see the Acrobat and PDF Library API Overview and the Acrobat and PDF Library API Reference.
Deciding Which Acrobat SDK Technology to Use

This chapter describes general issues that factor into decisions about which Acrobat SDK technology to use. Subsequent chapters provide overview information about different application-specific tasks that you can do with the Acrobat SDK and discuss the capabilities and limitations of each technology in performing those tasks.

- Getting Started
- When to Use Interapplication Communication
- Deciding Between Plug-ins and JavaScript
- Samples Provided with the Acrobat SDK

Getting Started

You can develop software that integrates with Acrobat and Adobe Reader in three ways:

- Acrobat JavaScript
- Plug-ins
- Interapplication Communication (IAC)

Based on your application’s requirements, you choose the appropriate technologies for development. In some situations, the desired functionality is only available using one particular technology. In other situations, you can choose between two or more technologies. For example, you can add menu items to Acrobat using either JavaScript or a plug-in.

You can also use more than one technology in a single application or single document. For example, you can use both plug-ins and JavaScript to implement a particular functionality. Also, you can extend the functionality of intercommunication application interfaces by writing plug-ins that use core API objects not already part of the IAC support system (see “Using Plug-ins for Interapplication Communication” on page 27).
When to Use Interapplication Communication

To take advantage of Acrobat functionality from within an external application, you use Interapplication Communication (IAC).

Viewing PDF Documents from an External Application

If your Windows application only views a PDF document and does not need to edit it in any way, it is recommended that you use the PDF Browser Controls to view the document from your external VB or C# application. When you open a document for viewing using the PDF Browser Controls, the document is displayed in the application window. Acrobat toolbars are also displayed and can be used with no additional API calls. The toolbars can optionally be hidden. See “PDF Browser Controls” on page 26 or the Acrobat Interapplication Communication Overview for more details.

**NOTE:** Acrobat or Adobe Reader must be installed on the end-user’s computer for the PDF Browser Controls to function correctly.

You can also use the IAC API to open and view a PDF document. However, when you use the IAC API, no toolbars are displayed; you must place your own buttons with corresponding API calls for standard toolbar tasks such as printing and searching.

The following table summarizes some of the issues to consider when deciding whether to use the PDF Browser Controls or IAC APIs.

<table>
<thead>
<tr>
<th>TABLE 3.1 Comparing the PDF Browser Controls and IAC APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PDF Browser Controls</strong></td>
</tr>
<tr>
<td>● Acrobat toolbars can be displayed with no additional API calls.</td>
</tr>
<tr>
<td>● Supports only viewing, not editing, of PDF documents.</td>
</tr>
<tr>
<td><strong>IAC API</strong></td>
</tr>
<tr>
<td>● Supports editing PDF documents and manipulating Acrobat or Adobe Reader through a richer set of APIs than the PDF Browser Controls. Note that the set of APIs for Adobe Reader is limited unless the PDF document has rights enabled (see Chapter 10, “Rights-enabled PDF Documents”).</td>
</tr>
<tr>
<td>● Acrobat toolbars cannot be displayed.</td>
</tr>
</tbody>
</table>

Controlling Acrobat From an External Application

If you need to do more than just view a PDF document from your application, you can use either the IAC API or, if your application is written in VB, you can access Acrobat JavaScript using the JSObject.
Using IAC
Using the IAC API, you can:

● Get annotations, text and form data from a PDF document
● Search a PDF document
● Manipulate a PDF document, editing and adding content
● Control Acrobat remotely. You cannot remotely control Adobe Reader.

From the Acrobat Viewer layer, you can use the IAC API to display PDF files in your application’s windows. You can change the control you have over the application’s user interface and appearance of its window by either using a PD layer object, `PDPage`, or by using an AV layer object, `AVDoc`. `PDPage` gives you access to Acrobat’s rendering capabilities. For more control, you can create your application with `AVDoc` that can be used to display text annotations and active links in your application’s window.

From the PD layer, you can perform basic manipulations of PDF documents, such as deleting, moving or replacing pages, as well as changing annotation attributes. You can print PDF pages, select text, manipulate text, and create or delete thumbnails.

For more information on the IAC API, see “Interapplication Communication (IAC)” on page 25 and the Acrobat Interapplication Communication Reference.

You can extend the functionality of the IAC interfaces by writing plug-ins that use core API objects not already part of the IAC support system. For more information, see “Using Plug-ins for Interapplication Communication” on page 27 and the Acrobat Interapplication Communication Overview.

Executing Acrobat JavaScript
You can execute Acrobat JavaScript commands from the IAC API by using the IAC API `ExecuteThisScript` method of the `AcroForm` plug-in. On Windows, you can also use the `JSObject` to access the JavaScript object model through the Acrobat automation interface. For more information, see “Accessing the JSObject Interface” on page 27 and Programming Acrobat JavaScript Using Visual Basic. On Mac OS X, you can execute JavaScript scripts using `Do Script` in AppleScript.

Deciding Between Plug-ins and JavaScript
This section presents topics to consider as you decide whether to use plug-ins or JavaScript to implement a particular functionality.

The section contains the following topics:

● General Issues
● What You Can Do with Plug-ins That You Can’t Do with JavaScript
● What You Can Do with JavaScript That You Can’t Do with a Plug-in
**NOTE:** Subsequent chapters of this document discuss specific tasks you can do using the Acrobat SDK and which technologies you can use to perform those tasks.

### General Issues

The table below lists other general issues that are relevant to deciding between plug-ins and JavaScript.

<table>
<thead>
<tr>
<th><strong>TABLE 3.2</strong> Plug-ins vs. JavaScript - General Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope</strong></td>
</tr>
<tr>
<td>● A plug-in affects all PDF documents viewed by Acrobat.</td>
</tr>
<tr>
<td>● JavaScript can affect either a single document or all PDF documents in a particular folder.</td>
</tr>
<tr>
<td><strong>Installation/Distribution</strong></td>
</tr>
<tr>
<td>● Plug-ins must be placed in the Plug_ins folder or directory by an installer or by the user.</td>
</tr>
<tr>
<td>● Document-level JavaScripts are easier to distribute since they can be included directly within the PDF file and do not require an installer. Folder-level JavaScripts must be placed in the Acrobat application JavaScript folder or the user’s JavaScript folder.</td>
</tr>
<tr>
<td><strong>Low-level Access</strong></td>
</tr>
<tr>
<td>● Plug-ins can access and manipulate low-level objects in the PDF object model, such as the Cos layer.</td>
</tr>
<tr>
<td>● JavaScript scripts can only access a limited set AV and PD layer objects.</td>
</tr>
<tr>
<td><strong>Execution Speed</strong></td>
</tr>
<tr>
<td>● Plug-ins are compiled and loaded when Acrobat initializes.</td>
</tr>
<tr>
<td>● Execution of JavaScript code is generally slower than plug-in code because it is interpreted instead of compiled. The difference is noticeable only in very computation-intensive applications, such as a full-text search in a large PDF file.</td>
</tr>
<tr>
<td><strong>Ease of Implementation</strong></td>
</tr>
<tr>
<td>● Plug-ins are developed in C or C++ and are compiled and linked in the appropriate development environment. You must be careful to include all necessary header files for your application.</td>
</tr>
<tr>
<td>● JavaScript scripts are easier to write and implement since they are developed using the editor and debugger that come as part of Acrobat Professional. Developers can also use an external editor to create and edit JavaScripts.</td>
</tr>
<tr>
<td><strong>Cross-platform compatibility</strong></td>
</tr>
<tr>
<td>● Plug-ins must be built on different platforms to handle certain platform-specific issues.</td>
</tr>
<tr>
<td>● JavaScript is cross-platform compatible.</td>
</tr>
</tbody>
</table>
What You Can Do with Plug-ins That You Can’t Do with JavaScript

In general, plug-ins allow for more direct control over Acrobat than JavaScript. There is a much richer set of APIs that you can use from a plug-in.

The following can only be done using a plug-in; there is no equivalent JavaScript functionality:

**NOTE:** The following is not meant to be a complete list. For more detailed information, see the subsequent chapters of this document.

- Accessing Cos and other low-layer objects
- Creating new menus or toolbars
- Manipulating PDF content streams
- Creating new annotation or action types
- Modifying the ASFixed scale factor for large PDF file sizes.
- Accessing platform-specific services or events
- Getting and setting wireframe drawing mode

What You Can Do with JavaScript That You Can’t Do with a Plug-in

The following can be done easily using Acrobat JavaScript, but have no plug-in equivalent API.

**NOTE:** Some tasks, such as SOAP and web services, can be done with a plug-in by using low-level APIs. However, this is a very time-consuming approach and requires an in-depth knowledge of the Cos and other other low-level APIs.

**NOTE:** The following is not meant to be a complete list. For more detailed information, see the subsequent chapters of this document.

- Using SOAP and other web services
- Manipulating multimedia in PDF documents
- Setting up an automated email review workflow
- Describing the state model for the review
- Searching Acrobat Help
- Using Acrobat Security Policies
Samples Provided with the Acrobat SDK

The Acrobat SDK contains a large number of sample applications, plug-ins, and scripts to demonstrate how to use the SDK technologies.

**NOTE:** Samples are regularly being added to the Acrobat SDK. Please refer to http://partners.adobe.com/links/acrobat for the current complete list of samples available for the Acrobat SDK.

Plug-in Samples

There are two types of plug-in samples in the Acrobat SDK: complete plug-ins and plug-in snippets. Complete plug-ins are stand-alone plug-ins that you can compile and run in Acrobat. Snippets are small fragments of code demonstrating how to use a single API that are run using the SnippetRunner plug-in.

Complete Plug-in Samples

The “Starter” and “BasicPlugin” samples provide the most basic frameworks for developing an Acrobat plug-in. The “DebugWindowADM” sample demonstrates how to build tools to help you debug your plug-in.

Plug-in Snippets Using SnippetRunner

The SnippetRunner plug-in provides infrastructure to snippets. It is accessed from Advanced > Acrobat SDK> AcroSDK Snippets. Each snippet can determine what context should exist before it is called. For example, a snippet that reports the type of the current selection does not need to open a document and set a selection before reporting its type—it can assume the user has performed these tasks prior to invoking the snippet (the snippet should still check that a document has been opened, however). The snippet obtains the context from the utility code.

For more information, see the Guide to SDK Samples and the SnippetRunner Cookbook.
Acrobat JavaScript Samples

JavaScript samples are divided into six categories, each in its own folder in the Acrobat 7.0 SDK/JavaScriptSupport/Samples folder:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outside PDF</td>
<td>JavaScript files that live outside an Acrobat file — for example, the code used in JavaScript folders, the console, and batch processes.</td>
</tr>
<tr>
<td>Inside PDF</td>
<td>Scripts written in JavaScript that live inside a PDF file.</td>
</tr>
<tr>
<td>Multimedia</td>
<td>Scripts that demonstrate the use of multimedia in PDF documents.</td>
</tr>
<tr>
<td>ADBC</td>
<td>Scripts that demonstrate database connectivity using ADBC.</td>
</tr>
<tr>
<td>SOAP</td>
<td>Scripts that demonstrate how to use SOAP services using Acrobat JavaScript.</td>
</tr>
<tr>
<td>UI</td>
<td>Scripts that demonstrate how to manipulate the Acrobat user interface.</td>
</tr>
</tbody>
</table>

Interapplication Communication Samples

The Acrobat SDK includes samples that demonstrate IAC functionality in both the Macintosh and Windows development environments.
Deciding Which Acrobat SDK Technology to Use

Samples Provided with the Acrobat SDK
This chapter provides information about how to use the Acrobat SDK to create PDF documents.

This chapter consists of the following sections:

- Creating PDF Documents with Acrobat
- Creating PDF Documents from an Authoring Application
- Creating Tagged PDF Documents
- Creating PDF Documents Using Plug-ins or Acrobat JavaScript
- Creating PDF Documents with Acrobat

---

**Creating PDF Documents with Acrobat**

There are several ways that you can create PDF documents using software distributed as part of the Acrobat product.

**Acrobat PDFMaker**

Files created in many applications can be converted directly to Adobe PDF files without leaving the authoring application. The authoring application uses PDFMaker for the conversion and the resulting files are Adobe PDF files.

In Windows, the default Acrobat installation installs the PDFMaker feature for the following third-party applications:

- Microsoft Internet Explorer 6.0
- Microsoft Publisher 2002 and 2003
For the Mac OS, the default Acrobat installation installs the PDFMaker feature for the Professional, Standard, and Student and Teacher Editions of the following Microsoft applications:

- Microsoft Excel X (SR-1), 2004
- Microsoft PowerPoint X (SR-1), 2004
- Microsoft Word X (SR-1), 2004

When you install Acrobat using the default installation settings, the installer identifies third-party applications on your computer that support PDFMaker and installs the necessary PDFMaker files to enable those applications to convert files to PDF files. The PDFMaker files are also automatically installed if you install such a third-party application after installing Acrobat on Windows.

Adobe PDF Printer

In an application that does not have a PDFMaker, you can still create a PDF file using the Adobe PDF printer. Your source document is converted to PostScript and fed directly to Distiller for conversion to PDF, without manually starting Distiller. The current Distiller preference settings and Adobe PDF settings are used to convert the file.

Save As or Export

Use the Save As or Export command to create an Adobe PDF file from the current file. This method is available in authoring applications such as Adobe InDesign, Adobe Photoshop®, Adobe Illustrator®, and Adobe PageMaker. All the necessary components for creating Adobe PDF files are installed and configured automatically when you perform a typical installation.
Creating PDF Documents from an Authoring Application

Authoring applications can simplify the creation of PDF files by making the following steps appear seamless:

- Invoking Distiller through Apple events or DDE.
- “Printing” to a PostScript file.
- Automatically generating advanced Acrobat features using pdfmark.

This section consists of the following subsections:

- Acrobat Distiller
- Apple Event Support
- Distiller Automation on Windows Using OLE Automation
- Automated, Silent Creation of PDF Documents from Windows
- Choosing an Adobe PDF Settings File
- Automatically Generating Advanced Acrobat Features
- Inserting PostScript Code into a Job
- Attaching the Native Document to the PDF File
- Batch Processing with Distiller

Acrobat Distiller

Acrobat Distiller is essentially a PostScript interpreter that can be used to convert PostScript to PDF. Distiller is the PDF creation application intended for batch processing use and for the creation of PDF files containing high-end print publishing features such as OPI comments, CMYK color spaces, and spot colors. Distiller also has the ability to interpret PostScript extensions called pdfmarks and convert them to PDF objects such as links, bookmarks, optional content (OC), and annotations. See “Choosing an Adobe PDF Settings File” on page 41.

For more information on the Distiller program, see the following documents or http://partners.adobe.com/links/acrobat:

- Acrobat Distiller API Reference
- Acrobat Distiller Parameters
- pdfmark Reference Manual
Apple Event Support

The Macintosh version of Distiller supports Apple events. Apple events can be used from external applications written in programming languages such as C or from AppleScript. Because AppleScript is more straightforward, using it is the recommended way to use Apple events with Distiller.

Distiller supports the following Apple events:

- The Distill command distills one or more PostScript language files, creating a PDF file for each.
- The run command launches Distiller.
- The quit command quits Distiller.

Distiller Automation on Windows Using OLE Automation

For Windows, you can automate Distiller under Microsoft Windows through the Automation interface also known as OLE Automation (commonly referred to as ActiveX or COM). The Automation interface makes it easy to create and control a Distiller from any programming language that supports Automation. Distiller supports programming environments written for the following languages:

- Visual Basic
- Visual C++ with and without MFC

Distiller exposes one interface: PdfDistiller. This interface provides methods and properties to invoke Distiller and to customize Distiller properties. For example, you can:

- Specify that a particular settings file be used rather than the default settings file
- Apply security to the generated PDF file
- Specify whether to spool the job or execute it immediately

You can also use the Distiller API to check the status and errors associated with a Distiller job.

As it processes each job, Distiller fires several events to notify you of its progress. These events can be accessed using the PdfDistiller interface.

In addition, certain features of Distiller can be controlled directly through the PostScript language file (see “Choosing an Adobe PDF Settings File” on page 41). Authoring applications that control Distiller should also provide an interface to their users to allow control of font embedding and image compression as well as downsampling in the PDF file.

For more information, see the Acrobat Distiller API Reference.
Automated, Silent Creation of PDF Documents from Windows

In the Acrobat 7.0.5 SDK, there is a new set of APIs that allow you to automate the silent creation of PDF files from an external Windows application.

When you use the OLE Automation described above, the Distiller splashscreen and status are displayed even when the process is automated. With this API, there is no interaction with the user and no display on the screen; the printing is done “silently.”

This API allows you to specify the name and location of the output PDF. To do so, you must make modifications to the Windows registry. See the Distiller API Reference for more information on what registry modifications are required.

**NOTE:** This procedure involves editing the registry. Adobe Systems Incorporated does not provide support for editing the registry, which contains critical system and application information. It is recommended that you back up the registry before modifying it.

To check status and errors, use the Distiller API. For more information, see the Acrobat Distiller API Reference and the Acrobat SDK Samples Guide.

Choosing an Adobe PDF Settings File

You can control the Adobe PDF settings file through the Distiller user interface, or you can control Distiller programmatically through the various Distiller automation methods. For details on automation methods, see “Creating PDF Documents from an Authoring Application” on page 39 and the Acrobat Distiller API Reference.

At the user interface level, you can set Distiller parameters by selecting a file of predefined Adobe PDF settings, or by creating your own set of Adobe PDF settings. Each Distiller settings file defines compression, font embedding, color management, PDF/X-compliance, and many other properties for all PDF files that Distiller creates.

Distiller comes with several predefined Adobe PDF settings files, with settings defined for common scenarios. You choose which settings file to use as a parameter to the Distiller Automation API. For a complete list of settings, see the Acrobat Distiller API Reference.

Automatically Generating Advanced Acrobat Features

The most convenient place to generate advanced Acrobat features is in the authoring application itself, since this application knows the most about the structure of the document. Advanced PDF features are generated using the *pdfmark* PostScript operator. The authoring application must generate a PostScript language file that contains the appropriate pdfmark operators for the document structure desired. The resulting PostScript language file is converted into a PDF file via Distiller. In addition, Acrobat allows logical structure information to be added to a document. For instance, paragraphs in a page’s contents can be associated with a structural element representing a paragraph.

For more information, see “Creating Tagged PDF Documents” on page 43 and the *pdfmark* Reference Manual.
Inserting PostScript Code into a Job

Acrobat Distiller contains a PostScript 3 interpreter with two Distiller-specific operators, `setdistillerparams` and `currentdistillerparams`. An Adobe PDF settings file can be thought of as a Postscript file that contains `setdistillerparams` key-value pairs, followed by the PostScript `setpagedevice` operator with two key-value pairs. The Adobe PDF settings file is the first file processed by Distiller, but parameters can subsequently be set or changed within the PostScript job itself.

See Distiller Parameters for more information.

Attaching the Native Document to the PDF File

Another way that an authoring application can integrate with Acrobat or Adobe Reader is to allow the user access to the original authoring document through the Acrobat user interface. Through the use of private data in a PDF file, an authoring application can embed the entire authoring document as part of the PDF file that represents it. This way, not only can the resulting electronic document be viewed by anyone using Acrobat, it can also be edited by users who have the authoring application. You must write a plug-in for Acrobat n to allow users to embed and extract the authoring document. This plug-in would simply add the authoring document as a private data stream when embedding, and, when extracting, save the stream to a temporary file and invoke the authoring application.

NOTE: Acrobat and Adobe Reader ignore private data. Embedding authoring documents in PDF files greatly increases the size of the PDF file.

Using Links to Associate a Native Document with the PDF File

An association between the PDF file and the authoring document can also be maintained through the use of links in the PDF file. Links can be created that invoke files and their associated applications. If a document management system places such a link in the PDF file, users can invoke the original authoring document by executing the link.

Batch Processing with Distiller

Due to the built-in functionality of Distiller, you do not need to use the Distiller API to integrate it with your product. Distiller has the ability to watch directories over a network and to convert any PostScript files saved to those directories to PDF. It is also possible to set different job options for each directory so that one directory can be used, for example, for high-end color output, while the other can generate a more compressed file suitable for Web use. These features of Distiller are not supported by Acrobat Developer Support.

Check the help documentation packaged with the product or books by Adobe Press and other publishers for using Distiller through the user interface.

The Distiller API can be used to programmatically process files and set the output path and file names. The API is documented in the Acrobat Distiller API Reference. Also, the Acrobat Professional product can be used to set up watched folders.
NOTE: Server use of the Distiller software is not allowed. The End User License Agreement allows for use only on a single system. Access to, and use of, the Distiller software over a network is prohibited. The only exception is installation of the software. You are permitted to keep a copy of the software on a server so that users who have a license for the software can download and install it. A separate product, Acrobat Distiller Server, can be purchased or licensed from Adobe for server use.

Creating Tagged PDF Documents

PDF files are well known for representing the physical layout of a document; that is, the page markings that comprise the page contents. In addition, PDF versions 1.3 and beyond provide a mechanism for describing logical structure in PDF files. This includes information such as the organization of the document into chapters and sections, as well as figures, tables, and footnotes.

PDF 1.4 and Acrobat 5 introduced tagged PDF, which is a particular use of structured PDF that allows page content to be extracted and used for various purposes, including:

- Reflow of text and graphics
- Conversion to file formats such as HTML and XML
- Access for the visually impaired (see Chapter 14, “Accessibility”).

PDF Logical Structure

PDF logical structure is layered on top of a document’s page contents using a special markup language. HTML and XML use a similar layout for logical structure: text enclosed in a hierarchy of tags. In HTML, each component is wrapped with a set of tags that define its structure. For example, the text of a top-level header begins with a `<h1>` tag and ends with a `</h1>` tag. PDF provides similar constructs with its marked content operators.

In fact, HTML logical structure can be preserved in a PDF document. The Web Capture feature introduced in Acrobat 4.0 allows converting HTML to PDF. Such PDF may optionally contain structure information from the HTML data. Acrobat can generate bookmarks from this structure data.

The Structure Tree

Logical structure is independent of, though related to, the page content (that is, the actual marks on the page made by the marking operators).

In a PDF document, logical structure is represented by a tree of elements called a structure tree. There are pointers from the logical structure to the page contents, and vice versa. The structure tree provides additional capability to navigate, search, and extract data from PDF documents. By accessing a PDF document via its structure tree, for instance, you can obtain logically ordered content independently of the drawing order of the page contents.
Adding Structure Data to a PDF Document

Creating a tagged document directly from an authoring application is the best way to make PDF documents accessible to screen readers and reflow correctly on handheld devices.

If your PDF document was created without tags, Acrobat can add them. The tagging feature identifies most elements of a PDF document, including irregularly shaped columns, bulleted lists, captions that span columns, images that overlap text, and colored backgrounds.

From a plug-in, you can add structural information to any PDF file with the PDSEdit API. Once a file has logical structure, PDSEdit allows you to use it. For more information, see “Using Document Logical Structure” on page 51 and the Acrobat and PDF Library API Overview.

Using pdfmark to Add Structure Data to PDF

Advanced Acrobat features are generated using the pdfmark PostScript operator. The authoring application must generate a PostScript language file that contains the appropriate pdfmark operators for the document structure desired. This resulting PostScript language file is converted into a PDF file via Distiller.

The pdfmark operator is not a standard PostScript operator, but is provided as a PostScript language extension used by Distiller to describe features that are present in PDF, but not in standard PostScript. The pdfmark operator has been available beginning with Distiller 3.0, and has evolved with each release of the PDF specification.

**Note:** While the pdfmark operator provides for greater extensibility, it is not intended to define every feature that is present in PDF but not in standard PostScript.

For more information, see the pdfmark Reference Manual and the PDF Reference.
Creating PDF Documents Using Plug-ins or Acrobat JavaScript

You can use Acrobat and the Acrobat SDK to create a new, empty PDF file and to create a PDF file from supported file types.

- Creating an Empty PDF File
- Creating PDF Files from Multiple Files

Creating an Empty PDF File

Using either a plug-in or Acrobat JavaScript, you can dynamically create a new PDF file and modify its contents in an automated fashion. This can help make a document responsive to user input and enhance the workflow process.

Using Acrobat JavaScript

Acrobat JavaScript provides support for dynamic PDF file creation and content generation. This means that it is possible to dynamically create a new PDF file and modify its contents in an automated fashion.

Using a Plug-in

Using a plug-in, you can use the `PDDocCreate` API to create a new, empty PDF file. You can then use the `PDDocCreatePage` or `PDDocInsertPages` to add content to the created file.

Creating PDF Files from Multiple Files

It is possible to use either a plug-in, IAC or JavaScript to dynamically add content from other sources into a new PDF file. The sources may include files whose types conform to Multipurpose Internet Mail Extensions (MIME) type definitions.

Using Acrobat JavaScript

To import an external file into a PDF document, invoke the doc object’s `importDataObject` method. After doing this, it is possible to extract information from the data object for placement and presentation within the PDF document.

You can automate the insertion of multiple PDF files into a single PDF document through a series of calls to the doc object’s `insertPages` and `replacePages` methods.

Finally, to use a portion of the current document to create a new document, you can invoke the doc object’s `extractPages` method.

Using a Plug-in or IAC

You can use the `AVConversionToPDFHandler` API to extract data from other file types, and then bind the resulting PDF files into one PDF file using `PDDocInsertPages`. These APIs can also be conveniently executed directly from an IAC application.
Creating PDF Files Without Using Acrobat

This section briefly discusses other ways that you can create documents without using the Acrobat SDK. The subsequent sections are specific to creating PDF documents using the Acrobat SDK.

- **Acrobat Capture 3.0.x**
  Acrobat Capture 3.0.x is a separate Adobe product designed for the batch conversion of image files to PDF and creating searchable text PDF files from the resulting PDF files. For more information on the Acrobat Capture 3.0.x SDK, see partners.adobe.com/links/acrobat.

- **Adobe LiveCycle family of products**
  Adobe LiveCycle products enable businesses to dynamically generate customized PDF documents for more targeted and effective customer communications, allowing you to:
  - Flow XML data into richly formatted templates
  - Assemble and manipulate compound documents
  - Extract content from interactive Adobe PDF forms
  - Create customized, visually rich charts and graphics
  - Extend PDF documents using Adobe Reader extensions

Generating PDF Files Without Using Adobe Applications

Some developers have successfully developed the capability of generating PDF from their own applications without using Adobe products. Some of these developers have chosen to use Adobe’s PDF Library product to extend their own application. Others have built PDF-generation capability into their applications from scratch. This type of development is not supported by Adobe.

The best resource for building PDF-generation capability from scratch is the *PDF Reference*, which fully documents the PDF 1.6 specification.

The PDF file format is quite complex, and developing code to generate it requires a significant amount of development. In estimating development time, we suggest you look at the methods used to compress and encrypt the page content streams, as well as the requirements of the byte offset table at the end of the PDF file. The LZW compression technique is the subject of United States patent number 4,558,302 owned by the Unisys Corporation. Independent software vendors may be required to license this patent directly from Unisys to develop software using LZW. Further information can be obtained from Welch Licensing Department, Law Department, M/S C2SW1, Unisys Corporation, Blue Bell, Pennsylvania, 19424.
This chapter details how you can work with PDF documents using the Acrobat SDK. This chapter consists of the following sections:

- Automating Processes: Opening PDF Documents Silently
- Adding Navigation to PDF Documents
- Manipulating Pages of a PDF File
- Creating and Editing Page Content
- Using Document Logical Structure
- Modifying PDF Documents in Other Ways
- Adding Watermarks
- Spell-Checking
- Using Multimedia
- Interacting with OBDC Databases
- Printing PDF Files
- Working with Embedded Fonts
- Large Page Sizes in PDF Documents
- Wireframe Drawing
Automating Processes: Opening PDF Documents Silently

To open documents using automated processes, it may sometimes be necessary to suppress the warning indicating that the version number of the document is greater than the version of Acrobat or Reader being used.

To address this issue in the Windows registry for Acrobat, set `HKEY_CURRENT_USER\Software\Adobe\Adobe Acrobat\7.0\AVAlert\cCheckbox\idocNewerVersionWarning` to 1. For Reader, the registry entry is `HKEY_CURRENT_USER\Software\Adobe\Acrobat Reader\7.0\AVAlert\cCheckbox\idocNewerVersionWarning`.

**NOTE:** This procedure involves editing the registry. Adobe Systems Incorporated does not provide support for editing the registry, which contains critical system and application information. It is recommended that you back up the registry before modifying it.

To modify the property list entry for Acrobat on a Macintosh, use the property list file `~/library/preferences/com.adobe.acrobat.pro7.0.plist` and assign the value 1 to `<AVAlert><Checkbox><docNewerVersionWarning>`. For Reader, the property list file is `~/library/preferences/com.adobe.acrobat.reader7.0.plist`.

This warning can also be suppressed interactively by checking *Do not show this message again* in the warning dialog box that appears.

**NOTE:** In both cases, bear in mind that PDF files from future versions of Acrobat or other sources could conflict with the current version of Acrobat or Reader without displaying this warning, although Adobe Systems, Inc. makes every effort to avoid such backward compatibility issues.
Adding Navigation to PDF Documents

This section describes ways that you can add navigation to your PDF file using the Acrobat SDK.

- Adding and Managing Bookmarks
- Adding and Managing Thumbnails
- Adding and Managing Links

Adding and Managing Bookmarks

A bookmark corresponds to an outline object in a PDF document. A document outline consists of a tree-structured hierarchy of bookmarks that displays the document's structure, allowing the user to navigate interactively from one part of the document to another. Each bookmark has:

- A title that appears on screen
- An action that specifies what happens when the user clicks on the bookmark

Bookmarks can be either created interactively through the Acrobat user interface or generated programmatically. Typically, a user-created bookmark is used to move to another location in the current document, although other actions can be specified, such as opening a web link, opening a file, playing a sound or executing a JavaScript script.

Each bookmark in the bookmark tree structure has zero or more children that appear indented, and zero or more siblings that appear at the same indentation level. All bookmarks except the bookmark at the top level of the hierarchy have one parent, the bookmark under which it is indented. A bookmark is said to be open if its children are visible, and closed if they are not.

You can use Acrobat JavaScript, a plug-in, or IAC to get or set:

- The open attribute of a bookmark
- The text used for the bookmark's title
- The action that is invoked when the bookmark is selected

The object associated with each Acrobat SDK technology is shown below:

<table>
<thead>
<tr>
<th>JavaScript</th>
<th>Bookmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug-in</td>
<td>PDBookmark</td>
</tr>
<tr>
<td>IAC</td>
<td>AcroExch.PDBookmark</td>
</tr>
</tbody>
</table>
Adding and Managing Thumbnails

Page thumbnails are miniature previews of the pages in a document. You can use page thumbnails in Acrobat to jump quickly to a selected page and to adjust the view of the page.

When you move, copy, or delete a page thumbnail, you actually move, copy, or delete the corresponding page. You can embed page thumbnails in a PDF document so that they need not be redrawn every time you select the Pages tab. They can easily be removed and embedded again if necessary.

You can use the Acrobat SDK to:

- Create thumbnails in a PDF document
- Remove thumbnails from a PDF document

The object associated with each Acrobat SDK technology is shown below:

<table>
<thead>
<tr>
<th>JavaScript</th>
<th>doc.addThumbnails, doc.removeThumbnails</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug-in</td>
<td>PDThumb</td>
</tr>
<tr>
<td>IAC</td>
<td>AcroExch.PDDoc methods</td>
</tr>
</tbody>
</table>

Adding and Managing Links

Links, or hyperlinks, let users jump to other locations in the same document, to other electronic documents, or to Web sites. You can use links when you want to ensure that your reader has immediate access to related information. You can also use links to initiate actions.

The Acrobat SDK provides support for the addition, customization, or removal of links within PDF documents. These links may be used to access URLs, file attachments, or destinations within the document.

The objects associated with each Acrobat SDK technology is shown below:

<table>
<thead>
<tr>
<th>JavaScript</th>
<th>doc, Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug-in</td>
<td>PDLink</td>
</tr>
<tr>
<td>IAC</td>
<td>AcroExch.PDAnnot</td>
</tr>
</tbody>
</table>

The `doc` object contains methods for adding, retrieving, and removing links. This is used in conjunction with the `Link` object, which contains properties as well as a `setAction` method for customizing the appearance and behavior of a given link.

In addition, the Acrobat JavaScript `app` object contains a property called `openInPlace`, which can be used to specify whether cross-document links are opened in the same window or in a new one.
Using Actions for Special Effects

Thumbnails, bookmarks, links, and other objects have actions associated with them, and you may use Acrobat JavaScript to customize their behavior. For example, you can use them to display messages, jump to destinations in the same document or any other, open attachments, open Web pages, execute menu commands, or perform a variety of other tasks.

You can also customize these actions so that they change their appearance after the user has clicked on them. For a button, you can invoke its **highlight** property, which allows you to specify how the button will appear once it has been clicked.

### Manipulating Pages of a PDF File

You can use the Acrobat SDK to insert or remove pages from a PDF document. The following table shows the methods to use for each Acrobat SDK technology:

<table>
<thead>
<tr>
<th>Acrobat JavaScript doc object method</th>
<th>Core API method</th>
<th>IAC method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create an empty page in the current document</td>
<td><strong>newPage</strong></td>
<td>PDDocCreatePage</td>
</tr>
<tr>
<td>Insert pages from another document into the current document</td>
<td><strong>insertPages</strong></td>
<td>PDDocInsertPages</td>
</tr>
<tr>
<td>Move a page to another location in the same document</td>
<td><strong>movePage</strong></td>
<td>PDDocMovePage</td>
</tr>
<tr>
<td>Replace pages with pages from another document</td>
<td><strong>replacePages</strong></td>
<td>PDDocReplacePages</td>
</tr>
<tr>
<td>Delete pages from the current document</td>
<td><strong>deletePages</strong></td>
<td>PDDocDeletePages</td>
</tr>
</tbody>
</table>

**Note:** You can also access Acrobat JavaScript functionality using the **JSObject** from an external application. See “Accessing the JSObject Interface” on page 27.
Creating and Editing Page Content

Page content is a major component of a PDF file. It represents the visible marks on a page that are drawn by a set of PDF marking operators. The set of marking operators for a page is also referred to as a display list, since it is a list of marking operations that represent the displayed portion of a page. See the PDF Reference for an overview of page content streams and references to other chapters that describe the marking operators in detail.

PDFEdit API

The PDFEdit API provides easy access to PDF page contents. With PDFEdit, your plug-in can treat a page’s contents as a list of objects rather than manipulating the content stream’s marking operators.

PDFEdit is meant to be used in conjunction with the Acrobat PD layer and Cos layer methods for manipulating PDF documents. To use PDFEdit effectively, you should be familiar with PDF page marking operators and the PD layer of the core API, described in the Acrobat and PDF Library API Overview.

Acrobat Distiller and PDFWriter create documents from PostScript or as output from a printer driver. Non-PDFEdit methods in the core API allow displaying and printing documents, and provide the ability to rearrange pages and to add annotations. However, most of these manipulations are creation-centered, or only deal with objects at the page level and above. PDFEdit methods, on the other hand, allow your plug-in to deal with objects at the level of a page’s contents.

There is no Acrobat JavaScript equivalent to the PDFEdit API to allow you to manipulate page content.

For more information, see the Acrobat and PDF Library API Overview.

PDFEdit Samples

There are several “snippets” available from the SnippetRunner sample plug-in that demonstrate how to add data to the contents of a page. Using the Acrobat core API greatly simplifies modifying and creating PDF page contents. In addition, Acrobat updates the byte offset table and page resources to ensure that the PDF file is still readable after modification. This would be significantly more complex if this were attempted by parsing and rewriting the page contents without using the core API, and could result in an unreadable PDF file.

For more information, see the SnippetRunner Cookbook.
Using Document Logical Structure

As discussed in “Creating Tagged PDF Documents” on page 43, you can insert logical structure into a PDF document by creating a tagged PDF document.

The PDSEdit API provides the ability to add, modify and view this logical structure. For more information, see the Acrobat and PDF Library API Overview.

Navigating a PDF Document

PDSEdit methods allow you to navigate through a document according to its structure. Bookmarks made from structure can go to an individual paragraph or a whole section, rather than just to a point on a page. PDSEdit also allows searching within structure elements. For example, you can search for a word within elements of a certain type, such as headings. It can be used to move around a document, to analyze its content, and to traverse its hierarchical structure.

Extracting Data From a PDF Document

The PDSEdit API allows you to extract portions of pages according to their context, such as all of the headings or tables. The extracted data can be used in different ways, such as summarizing document information, importing the data into another document, or creating a new PDF document.

Adding Structure Data to a PDF Document

Authoring applications create documents that can be converted to PDF. When the document is converted to PDF and viewed, Acrobat does not automatically add structure to the document. You can add structural information to any PDF file with the PDSEdit API. Once a file has logical structure, you can use PDSEdit to access and modify that structure.

Using pdfmark to Add Structure Data to PDF

Authoring applications may add structure pdfmarks to the PostScript language code generated when a document is printed. When the Acrobat Distiller application creates a PDF file from such PostScript code, it generates structure information in the PDF file from the pdfmarks. This approach requires that either the authoring application to add structure pdfmarks to the PostScript code it generates, or that some other application generates the pdfmarks. See “Adding Structure Data to a PDF Document” on page 44 and the pdfmark Reference for more information.
Modifying PDF Documents in Other Ways

You can also use a plug-in or Acrobat JavaScript to modify a PDF document by:

- Cropping and rotating pages
- Numbering pages
- Adding headers and footers

For more information, see the Acrobat JavaScript Scripting Guide and the Acrobat and PDF Library API Overview.

Adding Watermarks

Acrobat JavaScript provides methods to create watermarks within a document, and place them in optional content groups (OCGs).

There are no high-level C APIs for adding watermarks. However, you can add watermarks using lower-level APIs. The Acrobat SDK contains a sample plug-in that adds a watermark. See the Guide to SDK Samples for more information.

Fixed Printing for Engineering Documents

To meet government requirements, it is often necessary that a watermark be included on any printed pages of large documents. To support this requirement, there is a new type of object in a PDF similar to an Optional Content Group. It is called a FixedPrint element and it prints at the same size and scale factor and in the same location on any printout (or export as Postscript) regardless of the settings used at print time.

The FixedPrint object is only used when printing or exporting to PostScript. It is not visible from within Acrobat.

The content of a FixedPrint is fixed to a region of a printed page. FixedPrint uses marked content so that any content is permitted. The placement coordinates can be either relative or absolute.

You can access FixedPrint using the improved header, footer and watermarking tools in the Acrobat user interface. There are no high-level plug-in APIs for manipulation FixedPrint objects. You can, however, manipulate FixedPrint from using PDEdit and Cos level APIs. There are no equivalent Acrobat JavaScript APIs.
Spell-Checking

Acrobat provides a Spelling plug-in that can scan a document for spelling errors. Using any of the Acrobat SDK technologies, you can do the following:

- Add or remove a dictionary from the list of available dictionaries
- Add or remove a spelling domain (search scope) from the Spell Check dialog box
- Add or remove a word in the user's dictionary
- Check the spelling of an individual word
- Ignore all occurrences of a word in a document when spell-checking
- Scan a text buffer and return the next word
- Set the document's dictionary search order
- Set the document's dictionary search order from an array of ISO 639-2 and 3166 language codes, allowing you to identify a dictionary by language rather than by name

The following additional functionality is available to plug-ins and external applications, but is not available using Acrobat JavaScript:

- Check a text object and optionally receive a callback for each change as the user interacts with the Spell Check dialog box
- Count the words in a text buffer that are contained in each of a set of dictionaries
- Create a new custom user dictionary and add it to the list of available dictionaries

Using Multimedia

Multimedia objects can be included in the content of PDF documents, as well as in annotations. You can only manipulate multimedia objects and players using JavaScript; you cannot use a plug-in.

Using Acrobat JavaScript, you can:

- Customize the settings, renditions, and events associated with media players
- Access and control the properties for all monitors connected to the system
- Add movie and sound clips
- Add and edit renditions
- Control rendition settings
- Set multimedia preferences that apply throughout a document

For more information, see the Acrobat SDK JavaScript Scripting Guide and the Acrobat SDK JavaScript Scripting Reference.
Interacting with OBDC Databases

Acrobat JavaScript provides an ODBC-compliant object model called Acrobat Database Connectivity (ADBC), which can be used in document-level scripts to connect to a database for the purposes of inserting new information, updating existing information, and deleting database entries. ADBC provides a simplified interface to ODBC, which it uses to establish a connection to a database and access its data, and supports the usage of SQL statements for data access, update, deletion, and retrieval.

To use ADBC, ODBC must be installed on a client machine running a Microsoft Windows operating system. Note that ADBC does not provide security measures with respect to database access; it is assumed that the database administrator will establish and maintain the security of all data.

Printing PDF Files

Using the Acrobat SDK, you can control the way that Acrobat, Adobe Reader or your external application prints PDF files.

Customizing and Automating Printing

Using any of the Acrobat technologies, you can customize the way that a PDF document is printed.

Printing from Acrobat JavaScript

Since printing involves sending pages to an output device, there are many options that can affect print quality. Acrobat JavaScript can be used to enhance and automate the use of these options in print production workflows, primarily through the use of the PrintParams object.

For a complete list of parameters, see the Acrobat JavaScript Scripting Guide.

Printing from a Plug-in

You can use AVDoc methods to customize and control how a PDF file is printed from Acrobat or Adobe Reader. Depending on the methods you use, a user interface is either invoked or not. The AVDocDoPrint method invokes the print user interface, where the user can then specify printing parameters.

The AVDocPrintPages method prints a document to the current printer using the current print settings and job settings with no user interface. You can specify a page range, a PostScript version and whether to shrink the pages to fit the printer.
To set more specific parameters for printing without a user interface, you must use the AVDocPrintPagesWithParams method. Among the parameters that you can set with this method are:

- The printer to be used
- Output file name (path and filename)
- Whether to embed the output in a bounding rectangle on the screen
- Whether to print the document as an image
- Whether to tile the output
- Output transparency level

**Printing from an External Application**

Using the IAC APIs, you can print a PDF file from an external application. On Windows, the IAC API’s AVDoc object has several methods for printing.

In AppleScript, the document object’s print pages method prints a specified range of pages without displaying a print dialog box.

For more information, see the Acrobat Interapplication Communication Overview.

---

**Working with Embedded Fonts**

This section discusses embedded font-related operations using the Acrobat SDK. Acrobat Distiller and the PDF Library add font embedding information to fonts that are embedded in PDF files. With the inclusion of this information, your code can determine how an embedded font can be used. The operations discussed in this section also largely apply to code used with the PDF Library SDK.

Acrobat plug-in developers can remove and embed fonts within an existing PDF document. They can also use fonts that are already embedded in a PDF document for preview and printing, as well as editing. However, allowing editing using embedded fonts is not recommended, and in some cases it is impractical. For example, CJK fonts potentially include thousands of glyphs, so applications must subset these fonts when embedding them in a PDF file. This precludes embedded CJK fonts from being used for editing by a plug-in.

PDF Library users can perform all of the above operations using an existing PDF document, or create a PDF document from scratch that includes embedded fonts. Creating a document from scratch cannot be performed by a plug-in, but it can be done by using PDF library calls from within a compiled application that includes the PDF Library.
Large Page Sizes in PDF Documents

In previous versions of Acrobat, the maximum size of a PDF document was 200 by 200 inches due to the internal use of ASFixed variables for all coordinates.

In Acrobat 7, pages in a PDF document now have a scaling factor associated with them that allows the same number of ASFixed points to represent feet, yards, meters, hectares or other measurements.

The value of this scaling factor is per page and can be retrieved in a plug-in using the PDPage GetUserUnitSize API. There are no Acrobat JavaScript APIs to access the unit size information.

Wireframe Drawing

In order to improve the speed of drawing large and complex documents, Acrobat has a new mode that only draws the strokes of objects as simple, one-unit lines as seen in the figure below. It does not affect filled shapes or text.

You can get and set wireframe mode from a plug-in using the following APIs:

- AVPageViewGetWireframeDrawing
- AVPageViewSetWireframeDrawing

There are no Acrobat JavaScript APIs for wireframe drawings.
Modifying the User Interface

This section describes how to change the Acrobat or Adobe Reader user interface using the Acrobat SDK.

This chapter consists of the following sections:

- Adding or Removing Menus and Menu Items
- Modifying the Toolbar
- Using the Adobe Dialog Manager (ADM)
- Customizing Acrobat Help

Adding or Removing Menus and Menu Items

You can use the Acrobat SDK to manipulate menu items, menus and menu bars. For complete information on what functionality is available with each SDK technology, see the Acrobat and PDF Library API Reference, the Acrobat JavaScript Scripting Reference, or the Acrobat Interapplication Communication Reference.

Adding Menu Items

You can use a plug-in or Acrobat JavaScript to add or remove menu items or submenus to an existing menu. From JavaScript, use the app object’s AddMenuItem, HideMenuItem, and AddSubMenu methods. Plug-ins can access the methods of the AVMenu and AVMenuItem objects.

With IAC OLE automation, you can remove a menu item using the App.MenuItemRemove method. However, you cannot use IAC to add a menu item.

Menu items added by plug-ins can have shortcuts (keyboard accelerators). Acrobat and Adobe Reader do not ensure that plug-ins add unique shortcuts, but it is possible for a plug-in to check which shortcuts are already in use before adding its own. Note that the only way to ensure there are no shortcut conflicts is for all plug-ins to check for conflicts before adding their own shortcuts.

You are encouraged to have your plug-in add its menu items to the Tools menu. When it is launched, Acrobat or Adobe Reader automatically adds this menu, as well as the About Plug-ins and Plug-in Help menus (see “Placing Plug-in Help Files in a Standard Location” on page 64). After Acrobat or Adobe Reader loads all plug-ins, it checks these three menus and removes any that are empty.
Modifying the User Interface

Modifying the Toolbar

Adobe keeps a registry of plug-in menu item names to help avoid conflicts between plug-ins. For more information on registering and using plug-in prefixes, see the Acrobat SDK Plug-In Guide.

Adding a New Menu

To create a new menu, use the AVMenuNew method from a plug-in. Add a new (or an existing) menu to a menu bar using the AVMenuBarAddMenu method.

You cannot create menus or add menus to menu bars using Acrobat JavaScript or IAC.

Modifying the Toolbar

You can use the Acrobat SDK to manipulate tool buttons and toolbars. For complete information on what functionality is available with each SDK technology, see the Acrobat and PDF Library API Reference, the Acrobat JavaScript Scripting Reference, or the Acrobat Interapplication Communication Reference.

Adding Items to an Existing Toolbar

You can add or remove buttons to the toolbar, although the size and resolution of the user’s monitor can limit the number of tool buttons that can be added.

From a plug-in, use the methods of the AVToolBar and AVToolButton objects. From Acrobat JavaScript, use the app object’s addToolButton and removeToolButton methods.

With IAC OLE automation, you can remove a menu item using the App.MenuItemRemove method. However, you cannot use IAC to add a menu item.

Creating a New Toolbar

Plug-ins also can create new toolbars, called flyouts, that can be attached to buttons on the main toolbar. The selection tools, for example, are all on a flyout. As of Acrobat 4.0, it is no longer true that all tool buttons are located on the main toolbar; some may be located on a flyout.

You cannot create a flyout with Acrobat JavaScript or IAC.
Using the Adobe Dialog Manager (ADM)

The Adobe Dialog Manager (ADM) is a cross-platform API for implementing dialog interfaces for Adobe applications such as Acrobat, Adobe Reader, Photoshop, Illustrator, and After Effects. This document describes ADM structures and how to access them. You should already be familiar with the concept of dialogs and dialog items.

For complete information about ADM, see the Adobe Dialog Manager Programmer's Guide and Reference.

This section covers the following topics:

- ADM Controls
- ADM Objects
- Using ADM from Acrobat JavaScript
- Using the ADM API from a Plug-in

ADM Controls

ADM enables developers to create and manage cross-platform dialogs. Two types of dialogs are supported: modal dialogs and modeless dialogs. The latter dialogs “float” over the host application windows, while the former are displayed and disappear upon conclusion of the user input.

With a modal dialog, a user cannot work elsewhere in the application until the dialog is closed. In both cases, ADM supports a wide variety of control types, including basic ones such as buttons and text, and more complicated types such as lists and hierarchy lists.

In addition to providing a wide array of custom and standard user interface elements, ADM also provides some very useful features, such as tab palettes, docking palettes, and automatically tracking and displaying the correct selection in grouped radio buttons. Finally, ADM provides a consistent Adobe interface and “look and feel.”

The following types of controls are available using ADM:

- Push button
- Check box
- Radio button
- List box
- Hierarchical list box
- Static text box
- Editable text box
- Popup control
- OK button
- OK and Cancel button
- OK, Cancel, and Other button

You can also create containers or frames to work with sets of controls. The following is an example of a cluster of several items:

![Sample Dialog](image)

**ADM Objects**

ADM user interfaces are built from ADM user interface objects. These objects include the dialog windows (dialog objects) and the dialog items (item objects) within the windows.

![ADM Objects](image)

**Figure 6.1 ADM Objects**

A plug-in or application using ADM has access to standard ADM dialog types (modal and non-modal) and items (buttons and other user interface controls). The user interface can be built using code or by using resource definitions. For instance, standard platform resources
can be used to define the layout of the user interface objects. All objects have properties and events that determine their default behavior and allow them to be modified or extended. These can also be set in code or using a resource.

ADM has an object-oriented design even though its interfaces are exported as procedural C functions. This is important since many of the properties, behaviors, and callback functions of the various types of ADM user interface objects (dialogs or dialog items) are the same. Understanding the fundamentals of managing one type of ADM user interface object results in understanding how to manipulate other ADM objects as well.

Some ADM objects require additional support functions or properties. A window object, for instance, has functions to perform operations such as setting the minimum and maximum window size. ADM edit text items have additional functions to support properties such as justification and numeric precision.

**Using ADM from Acrobat JavaScript**

Acrobat JavaScript provides a convenient interface to ADM through which you may implement modal dialogs containing list objects and other controls normally included in graphical user interfaces.

The app object’s `execDialog` method is the Acrobat JavaScript method used to execute ADM dialogs. The method requires a single parameter: an object literal known as a dialog descriptor, which contains properties and method handlers for all the dialog elements.

For more information, see the Acrobat JavaScript Scripting Guide.

**Using the ADM API from a Plug-in**

Acrobat exposes ADM functionality to plug-in developers. This allows developers to use a single code base to implement dialog behavior on both Macintosh and Windows.

**Note:** You do not have to write platform-specific code to support Macintosh and Windows. However, it is still necessary to create platform-specific dialog resources using the appropriate tools on each platform.

The Acrobat SDK provides header files to simplify the use of ADM. In addition, several samples and snippets in the Acrobat SDK illustrate the use of ADM.

ADM contains a series of suites, each of which implements different functionality. Each suite has its own header file that declares its functions, data types, constants, and so on.

For more information, see the Adobe Dialog Manager Programmer’s Guide and Reference.

**Macintosh Issues**

Carbon is a set of programming interfaces that can be used to build Mac OS X applications that also run on Mac OS 8 and 9 (versions 8.1 and later). It includes a set of header files and a library called CarbonLib. These interfaces include most of the functions commonly used by Macintosh developers. By developing with these interfaces and linking with CarbonLib
(instead of previous libraries such as InterfaceLib), a “carbonized” executable is produced which can run under Macintosh OS X (in native mode) as well as Macintosh OS 8/9.

The Acrobat 5.0 and above products are carbonized. In order for a plug-in to use ADM, it must also be carbonized.

**NOTE:** Plug-ins which do not use ADM will still work on versions other than Mac OS X even if they are not carbonized.

To use ADM with Acrobat on a Macintosh, you must also use a `plist`.

For more information, see the Adobe Dialog Manager Programmer’s Guide and Reference.

---

**Customizing Acrobat Help**

You can manipulate specific parts of the Acrobat help system using a plug-in. There is no equivalent functionality from Acrobat JavaScript or IAC.

**Controlling the “About” Box and Splash Screen**

Plug-ins can set values in the preferences file (using the `AVAppSetPreference` method) to prevent the Acrobat or Adobe Reader “About” box and/or splash screen from appearing before displaying the first document. These changes take effect the next time Acrobat or Adobe Reader is launched.

**Placing Plug-in Help Files in a Standard Location**

The `Help` directory that accompanies Acrobat or Adobe Reader provides a standardized location for your plug-in help files. `About Acrobat Plug-Ins` is a standard menu item in the `Help` menu. This menu item contains a submenu. You are encouraged to have your plug-in add a menu item to the submenu to bring up its own “About” box.

You cannot use Acrobat JavaScript to set values in the preferences file.

You can place a help file either in the `Help` directory or in a subdirectory of the `Help` directory. If, for example, your plug-in is localized for Japanese, you might want to place its Japanese help file in a `Help_JPN` subdirectory. To aid in opening locale-specific help files, the core API provides the `AVAppOpenHelpFileWithParams` method. There is no equivalent Acrobat JavaScript functionality.

For more information, see the Acrobat and PDF Library API Reference.
Customizing the How-To Panel

The core API provides several methods to customize the How-To panel from a plug-in. Use `AVAppRegisterHowToPanel` to register an HTML or XML file to appear in the How-To panel.

`AVAppAutoShowHowToPanel` opens the How-To panel and fills it with the specified panel's contents, if that panel's AutoShow attribute is `true`. You can toggle the AutoShow attribute using `AVAppSetHowToPanelAutoShow`.

There is no equivalent Acrobat JavaScript functionality. For more information, see the Acrobat and PDF Library API Reference.
This chapter describes how to use the Acrobat SDK to assist with online collaboration workflows and annotations.

This chapter includes the following sections:

- Using Online Collaboration
- Using Annotations in Document Workflows
- Using Stamps in Approval Workflows

## Using Online Collaboration

Acrobat provides a platform for the development of a variety of collaborative review systems to fit various needs. A complete, integrated Acrobat and Adobe PDF online collaboration solution includes the following major elements:

- Adobe PDF documents created from source documents authored in various word processing, page layout, and other applications.
- A Web browser environment, running the Adobe Acrobat plug-in, which allows users to view a PDF document on a network folder or a Web server, add comments to it, and work online with comments from the whole workgroup.
- A central, online comment repository for collecting and storing reviewer comments separate from the associated PDF document being reviewed.
- Optionally, depending on the complexity of the solution required, a user interface to the review and comment workflow, potentially including automated routing and notification, implemented by the solution developer using server-side applications.
- A Review Management feature which allows users to invite new reviewers, send reminders, and go back online after working offline.

This online collaboration process allows multiple users to comment on a single document and to view comments from others in real time. Also, they can comment on a document at their own convenience and upload their comments at a later time with the offline option. For all repository types, commenting can only occur when the user opens the PDF file within a Web browser.

For more information about setting up online collaboration using Acrobat, see *Acrobat Online Collaboration: Setup and Administration*. 
How FDF Files Are Used

FDF (Forms Data Format) is an Acrobat-specific format used to store form data and other information from a PDF file. An FDF file that is created during a comment upload contains only the reviewer’s comments and a reference to the reviewed PDF file. Consequently, the FDF file is typically more compact than the corresponding PDF file, and does not need to be stored under the same parent directory or even on the same machine. For more information about FDF, see the PDF Reference.

FDF files can also contain JavaScript code that can be used to automatically configure the server preferences. For example, a user can navigate to an FDF file on a server and double-click on it. Acrobat then executes the JavaScript code in the file, which automatically configures the server preferences for the user.

Online Collaboration and Acrobat JavaScript

For online collaboration, Acrobat JavaScript provides several important capabilities:

- **Interface to a database** — Acrobat JavaScript provides objects to interact with a database via ADBC (Acrobat Database Connectivity). Using JavaScript, you can read values from a database, update or insert new data, and delete information. See “Interacting with OBDC Databases” on page 54 for more information.

- **Set the comment repository preference** — To use Acrobat’s collaborative review capabilities, you must specify the type of comment repository to be used, as well as its location. Those preferences can be set manually, or with Acrobat JavaScript placed in either the PDF document or its associated FDF file.

- **Interface to a web service** — Acrobat JavaScript provides objects to interact with a web service via SOAP, which is an XML-based transport protocol used for exchanging information. Because it is the developer’s choice as to which operations are performed by a given remote procedure, the Acrobat JavaScript SOAP support enables a more customized collaboration workflow. For more information, see “SOAP and Web Services” on page 74.

Extensibility

You can extend the Acrobat Online Collaboration feature in several ways:

- You can write JavaScript code to customize your review process and comment handling.

- You can integrate Acrobat’s Online Collaboration feature into a larger document management system.

- You can use Acrobat JavaScript to add additional annotations.

- Through the use of the Acrobat JavaScript SOAP object, you can customize the way that Acrobat handles annotations.

For more information, see the Acrobat JavaScript Scripting Guide and the Acrobat JavaScript Scripting Reference.
Using Annotations in Document Workflows

An annotation associates an object such as a note, sound, or movie with a location on a page of a PDF document, or provides a way to interact with the user via the mouse and keyboard. PDF includes a wide variety of standard annotation types, described in detail in the PDF 1.6 Reference.

Many of the standard annotation types can be displayed in either the open or the closed state. When closed, they appear on the page in some distinctive form depending on the specific annotation type, such as an icon, a box, or a rubber stamp. When the user activates the annotation by clicking it with the mouse, it exhibits its associated object, such as opening a pop-up window displaying a text note or playing a sound or a movie.

You can access and manipulate annotation data from both a plug-in and Acrobat JavaScript. However, you can only create new annotation types using a plug-in or the IAC API; you cannot create new annotation types from JavaScript.

Annotations and JavaScript

You can set, modify and access annotation information using the Acrobat JavaScript annot object. However, you cannot create a new annotation type from JavaScript. To do this, you must use a plug-in or the IAC API.

Using the annot object, you can:

- Add note comments
- Make text edits
- Highlight, cross out, or underline text
- Add or delete custom stamps (see “Using Stamps in Approval Workflows” on page 71)
- Add comments in a text box
- Add attachments
- Spell check in comments and forms (see “Using Stamps in Approval Workflows” on page 71)
- Add commenting preferences
- Change colors, icons, and other comment properties
Annotations and Plug-ins

The PDAnnot class is the abstract superclass for all annotations (see the PDF Reference). Acrobat and Adobe Reader have two built-in annotation classes: PDTextAnnot and PDLinkAnnot. Plug-ins can add movie, Widget (form field), and other annotation types. You can define new annotation subtypes by creating new annotation handlers (see “Creating New Annotation Types” on page 70). There are no objects of type PDAnnot, but your plug-in can use PDAnnot methods on any subclass of PDAnnot.

The Acrobat SDK provides three useful macros to cast among PDAnnot and its text annotation and link annotation subclasses. These are:

- CastToPDAnnot
- CastToPDTextAnnot
- CastToPDLinkAnnot

The PDAnnot methods include:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDAnnotGetRect</td>
<td>Gets an annotation's size and location.</td>
</tr>
<tr>
<td>PDAnnotGetSubtype</td>
<td>Gets an annotation's subtype.</td>
</tr>
<tr>
<td>PDAnnotIsValid</td>
<td>Tests whether an annotation is valid.</td>
</tr>
</tbody>
</table>

See the Acrobat and PDF Library API Reference for more information.

Annotations and the IAC API

You can use the AcroExch.PDAnnot object in the IAC API to set, modify, and access annotation information from external applications.

Creating New Annotation Types

PDF supports many standard annotation types. Your plug-in can create new annotation types, including any data they need. For example, a custom annotation type might allow a user to draw (not just type) in an annotation, provide support for multiple fonts or text styles, or support annotations that can only be viewed by specific users.

Support for new annotation types can be added by defining and registering an annotation handler. The Acrobat Movie plug-in, for example, uses this to support video annotations.

To add a new annotation type, a plug-in must provide a set of callbacks, specify them in the AVAnnotHandler structure, and register them with AVAppRegisterAnnotHandler. For more information, see the Acrobat and PDF Library API Overview.
Using Stamps in Approval Workflows

Use the **Stamp** tool to apply a stamp, dynamic stamp, or custom stamp to a PDF document. A stamp can be created from a set of predefined stamps, dynamically created using system and identity information, or customized from PDF files or common graphic formats.

Using Acrobat JavaScript or a plug-in, you can apply any predefined or custom stamp. However, you cannot create custom stamps using the Acrobat SDK. There is also no Acrobat SDK support for Hanko and Inkan stamps to support Japanese workflows. These types of stamps can only be created using the Acrobat user interface.
Using Online Collaboration and Annotations

Using Stamps in Approval Workflows
The Adobe XML architecture offers enterprises a step-by-step migration from manual, paper-based workflows to streamlined, automated processes that accelerate the flow of business-critical information between employees, customers, suppliers, and constituents. This chapter describes the central role that Acrobat and the Acrobat SDK play in XML-based workflows.

This chapter consists of the following sections:

- The Adobe XML Architecture
- Converting PDF Documents to XML Format
- Managing XML-based Information

---

### The Adobe XML Architecture

The Adobe XML architecture leverages the capabilities of XML and PDF to support a variety of business applications, while offering connectivity to systems through a variety of industry-standard and Adobe technologies.
XML Forms

The Adobe XML Forms model uses a Document Object Model (DOM) architecture to manage the components that comprise a form. These include the base template, the form itself, and the data contained within the form fields. In addition, all calculations, validations, and formatting are specified and managed within the DOM and XML processes.

Static XML forms were supported in Acrobat 6, and dynamic XML forms are now supported in Acrobat 7. Both types are created using Adobe Designer. A static form presents a fixed set of text, graphics, and field areas at all times. Dynamic XML forms are created by dividing a form into a series of subforms and repeating subforms. They support dynamically changing fields that can grow or shrink based on content, variable-size rows and tables, and intelligent data import/export features.

For more information, see Chapter 9, “Forms and the Acrobat SDK.”

XML Templates

Acrobat JavaScript defines a Template object that supports interactive form architectures. In this context, a Template is a named page within a PDF document that provides a convenient format to automatically generate and manipulate a large number of form fields. These pages contain visibility settings, and can be used to spawn new pages containing identical sets of form controls to those defined within the template.

For more information, see “XML Form Templates” on page 86.

Extensible Metadata Platform (XMP)

Adobe's Extensible Metadata Platform (XMP) is a labeling technology that allows you to embed metadata into the file itself. With XMP, desktop applications and back-end publishing systems use a common method for capturing, sharing, and leveraging metadata.

For more information, see Chapter 12, “Working with Metadata.”

SOAP and Web Services

Most current models of web services use the Extensible Markup Language (XML), which facilitates interoperability by permitting the exchange of information between applications and platforms in a text-based manner. The Simple Object Access Protocol (SOAP) is an XML-based messaging protocol used as a medium for information and instruction exchange between web services. Information sent using the SOAP standard is placed in a file called a SOAP envelope. The Web Services Description Language (WSDL) is an XML-based standard for describing web services and their capabilities. Adobe Acrobat's support for these standards makes it possible for PDF documents to locate services, understand their capabilities, and interact with them.

In order to guarantee interoperability between various platforms and applications, it is necessary that exchanged information adhere to a standardized format for data types. To
address this need, the XML Schema Definition Language (XSD) provides a standard set of definitions for the various data types that can be exchanged in SOAP envelopes.

Acrobat 7 provides support for the SOAP 1.1 and 1.2 standards to enable PDF forms to communicate with web services. This makes it possible to include both SOAP header and body information, send binary data more efficiently, use document/literal encoding, and facilitate authenticated or encrypted communications. In addition, it also provides the ability to locate network services using DNS Service Discovery. All of this enables the integration of PDF files into existing workflows by binding Adobe XML forms to schemas, databases, and web services. These workflows include the ability to share comments remotely or invoke web services through form field actions.

For more information, see the Acrobat JavaScript Scripting Guide.

DNS Service Discovery

If the exact URL for a given service is not known, but it is available locally because it has been published using DNS Service Discovery (DNS-SD), Acrobat JavaScript provides methods to locate the service on the network and bind to it for communications.

For more information, see the Acrobat JavaScript Scripting Guide.

Workflow Applications

A SOAP-based collaboration server can be used to share comments remotely via a Web-based comment repository. Through the DNS Service Discovery support, it is possible to enable dynamic discovery of collaboration servers, initiation workflows, and RSS feeds that can provide customized user interfaces, via XML, directly inside of Acrobat 7.

In addition, it is possible to deploy Web-based JavaScripts that always maintain the most updated information and processes, and to connect to those scripts via form field actions that invoke web services.

For more information, see the Acrobat JavaScript Scripting Guide and Acrobat Online Collaboration: Setup and Administration.
Converting PDF Documents to XML Format

Because XML representation is the basis for the exchange of information with Web Services and enterprise infrastructures, it is often useful to convert your PDF documents into XML format.

Converting to XML Format Using Acrobat JavaScript

It is a straightforward process to do this using the `doc` object’s `saveAs` method, which not only performs the conversion to XML, but also to a number of other formats.

To convert your PDF document to a given format, you must determine the device-independent path to which you will save your file, and the conversion ID used to save in the desired format. For XML, the conversion ID is "com.adobe.acrobat.xml-1-00".

For more information, see the Acrobat JavaScript Scripting Guide and the Acrobat JavaScript Scripting Reference.

The SaveAsXML Plug-in

SaveAsXML is a plug-in for Acrobat which extends the “Save as type” choices in the Save As dialog box to allow a Tagged PDF document to be saved as a number of XML, HTML, or similar text-based formats.

Mapping Tables are used to control the conversion process for the SaveAsXML feature. The Mapping Tables are a script of hierarchically-organized directives written in a custom language defined in XML syntax. This allows developers to create custom Mapping Tables for formats other than those provided in this package.

Overview of the SaveAsXML Process

When the SaveAsXML plug-in registers itself with Acrobat, it inspects the set of XML files in the MappingTables folder to determine the number of conversion services that are available.

The MappingTables folder must be found in the same folder as the `SaveAsXML.api` file. Files in this folder are the only ones that are inspected as potential conversion services supported by the plug-in. This folder may not contain any files with the `.xml` extension that are not Mapping Tables.

If the registration process finds the Root element and its menu-name attribute, it adds the menu name to the list of file format choices available in the Save As dialog box. (The menu name must be unique in order to avoid confusion by similarly identified entries in the Save As dialog box’s file formats.)

For more complete examples of the usage of these directives, see the Mapping Tables distributed with SaveAsXML. Every directive that is currently supported has been used in one or more of these tables.
For more information, including a sample Mapping Table that demonstrates the basic operation of the SaveAsXML processing, see *Using the SaveAsXML Plug-in*.

---

**Managing XML-based Information**

Acrobat JavaScript provides support for XML-based information generated within workflows by providing an `XML` object, which represents an XML document tree that can be manipulated via the XFA Data DOM. (For example, it is possible to apply an XSL transformation to a node and its children.)

For more information, see the *Acrobat JavaScript Scripting Guide*. 
You can use the Acrobat SDK to extend the functionality of Acrobat Forms and XML forms created in Adobe Designer. This chapter presents basic concepts about forms and discusses how you can use the technologies of the Acrobat SDK to manipulate forms.

This chapter covers the following topics:

- Types of Forms
- Guidelines for Creating a New Form
- Acrobat Forms and the Acrobat SDK
- XML Forms and the Acrobat SDK
- Automatically Populating Forms
- Creating Web-Ready Forms
- Punchcard/Barcode Form Fields
- Collecting Data from Completed Forms
- XML Form Templates

### Types of Forms

In previous versions of Acrobat, PDF forms were generally Acrobat forms. You could create and manipulate Acrobat forms and retrieve form data using Acrobat JavaScript or, using the Forms API, from plug-ins and external applications.

In Acrobat 7.0, there is increased support for XML Forms, which are based on the XML Forms Architecture (XFA) specification. It is strongly recommended that new forms development be done using XML forms rather than Acrobat forms.

### XML Forms

A form is derived from a form design that you create using Adobe LiveCycle Designer. The form design specifies a set of layout, presentation rules, and data capture rules, including calculating values based on user input. The rules are applied whenever a form is filled with data. How the form is filled with data depends on the type of form. For information about creating forms, see the Adobe LiveCycle Designer documentation.

Using Adobe LiveCycle Designer, you can create the following types of forms:

- **Interactive** - An interactive form design produces a form that can be filled in online. Users can open the form in a browser or Adobe Reader and enter information into the form's fields. An interactive form can include buttons or commands for common tasks,
such as saving data to a file or printing. It can also include selection lists, calculations, and validations.

- **Static** - A static form design presents a fixed layout of text, graphics, and field areas. The resulting form retains its exact layout regardless of the amount of data merged into it. A static form design can also be designed to have interactive capabilities.

- **Dynamic** - A dynamic form design produces a form that has the ability to grow or shrink according to the amount of data being merged into it. A dynamic form design can also be designed to have interactive capabilities.

XML forms can be understood in terms of two major components: templates and content. The templates define presentation, calculation, and interaction rules, and are based on XML. Content is the static or dynamic data, stored in the document, that is bound to the templates.

For dynamic XML forms, the content is defined after binding to a template. This means that the following are possible:

- Fields can be moved or resized.
- Fields can automatically grow or shrink according to the amount of text added or removed.
- As a field grows, it can span multiple pages.
- Repeating subforms can be spawned as needed, and page contents will shift accordingly.
- Elements on the page are shown or hidden as needed.

To take advantage of the rich forms functionality offered by the XFA plug-in, use Adobe Designer to create or edit the templates and save your forms in the XML Data Package format (XDP) or as a PDF document. Use XDP if the data is processed by server applications, and PDF if the data is processed by Acrobat.

**Acrobat Forms**

With Acrobat forms, you can author form fields and retrieve data from those form fields. For Adobe Reader, the Forms plug-in does not allow form authoring, but allows users to fill in data and print Acrobat forms. The Adobe Reader Forms plug-in also does not allow users to save data to the local hard disk. Both Acrobat and Adobe Reader allow Web designers to send data from the form back to a Web server.
Forms Workflows

There are three basic workflows for forms:

**Fill and Print**
Forms are filled in on the screen and then printed out. They are then returned by traditional methods such as fax or postal mail.

**Email Submit**
Forms contain a Submit button that enables the sending of an email message with an attached data file that contains only the form data. See “Collecting Data from Completed Forms” on page 85.

**Online Submit**
Forms submit form data to a web server much like forms on the Internet. The user must be online to submit your information. For more information, see “Collecting Data from Completed Forms” on page 85.

Elements of Acrobat Forms

Acrobat offers several form field types for building and creating forms:

- Button
- Check box
- Combo box
- Digital signature field
- Document message bar
- List box
- Radio button
- Text field

To customize the utility and behavior of the form, you can define Acrobat JavaScript methods to respond to events. For more information about form field types, see the Acrobat JavaScript Scripting Guide.
Guidelines for Creating a New Form

Creating a new form, whether using Adobe LiveCycle Designer or programmatically using Acrobat JavaScript, requires you to consider specific issues. For guidelines, see the Acrobat JavaScript Scripting Guide.

Your form design may have dynamically changing features such as the current date, as well as convenience options such as automatic generation of email messages. It may even have a dynamically changing appearance and layout which is responsive to user interactions.

Acrobat Forms and the Acrobat SDK

For Acrobat forms, there is a rich set of APIs that allows you to create and manipulate form fields, and to retrieve form data using Acrobat JavaScript, a plug-in or an external application.

Though you can manipulate form fields and form data from a plug-in, it is much quicker to develop Acrobat forms using Acrobat JavaScript.

Using the Acrobat SDK, you can:
- Create Adobe PDF forms from scratch or from a template
- Add or remove form fields
- Set form field properties
- Make forms web-ready (see "XML Form Templates" on page 86)
- Extract and export form data (see “Collecting Data from Completed Forms” on page 85)
- Make forms accessible (see Chapter 14, “Accessibility”)

Acrobat JavaScript for Acrobat Forms

You can extend the functionality of Acrobat forms through the application of Acrobat JavaScript.

Acrobat JavaScript can be integrated into your forms to enhance their interactive capabilities. You can extend the capability of your forms by using Acrobat JavaScript to:
- Automate formatting, calculations, and data validation
- Develop customized actions assigned to user events
- Interact with databases and Web services

Acrobat JavaScript provides a field object, which has properties that define the field’s type, appearance and data values.
The Forms API

The Forms plug-in for Acrobat allows plug-in developers to author Acrobat form fields. For Adobe Reader, the Forms plug-in does not allow form authoring, but allows users to fill in data and print Acrobat forms. In general, the Adobe Reader Forms plug-in also does not allow users to save data to the local hard disk. However, if the PDF document has additional usage rights, then it may be able to save the document or perform other functions. For more information, see Chapter 10, “Rights-enabled PDF Documents”.

Both Acrobat and Adobe Reader allow Web designers to send data from the form back to a Web server.

The Forms API uses the PDFField object, which is an opaque object representing a field in an Acrobat form.

A PDF document that contains an Acrobat form has an AcroForm entry in the document catalog dictionary, which contains an array of references to each of the root fields in the document.

The three most important properties of a field are its type, name, and value. Other properties specify the appearance of a field. Fields can be organized into a hierarchy, and other field properties associate it with its parent and children.

OLE Automation for Acrobat Forms

The Acrobat Forms plug-in has been enhanced to work as an Automation server in the Win32 environment. There is no equivalent support in a Mac environment.

OLE Automation is particularly useful for creating custom forms from an external application. Methods and properties are provided that allow you to programmatically associate actions and Acrobat JavaScript scripts with form fields. You can also add document-level JavaScript scripts.

A very important and useful feature of OLE Automation of Acrobat forms is the ExecuteThisJavaScript method of the AFormApp object. The JavaScript string passed to this method is immediately executed in the context of the current form. You can also use the ExecuteThisJavaScript method to execute JavaScript scripts from non-forms PDF documents.

Since the automation capabilities have been added to a plug-in, rather than an executable that can be directly launched, there are several steps you must perform to be able to access them from an Automation controller. See the Acrobat and PDF Library API Reference for complete details.
XML Forms and the Acrobat SDK

For XML forms, there are additional properties and methods available only from JavaScript that allow you to access XFA-specific objects. You cannot access these objects from a plug-in or external application.

In particular, Acrobat JavaScript provides access to the XFA appModel container, which provides properties and methods to query and manipulate the XML field hierarchy.

For more information on XML Forms, see the Acrobat JavaScript Scripting Guide or the Acrobat JavaScript Scripting Reference.

Automatically Populating Forms

You can populate forms from a database or using a standards-based network protocol such as SOAP. For more information, see “Interacting with OBDC Databases” on page 54 and “SOAP and Web Services” on page 74.

XML Forms are particularly well suited for populating forms from an external database.

Creating Web-Ready Forms

Both Acrobat and XML forms can be used in workflows that require the exchange of information over the Web. You can create forms that run in Web browsers, and can submit and retrieve information between the client and server by making a Submit button available in the form. The button can perform tasks similar to those of HTML scripting macros.

For more information, see the Acrobat JavaScript Scripting Guide.

Punchcard/Barcode Form Fields

With Adobe LiveCycle Designer, you can add interactive barcode form fields for various identification purposes, such as product inventory. The bars and characters of interactive barcode fields change to encrypt the data that is entered into other associated form fields. Users filling out a form with a barcode cannot manually enter data into the barcode form field.

You cannot read or modify barcode form fields using the Acrobat SDK.
Collecting Data from Completed Forms

XML and XDP Formats

You can save your form data in pure XML format, or, to take advantage of the rich functionality offered by the XFA plug-in, you can save your forms in the XML Data Package format (XDP).

The XDP format allows you to package units of PDF content within a surrounding XML container, and is thus considered an XML-based companion to PDF. The advantage of this format is that it provides XML parsers direct access to the XML form-data subassembly of the PDF document.

For more information, see the Acrobat JavaScript Scripting Guide.

FDF and XFDF Formats

Using Acrobat JavaScript, form data can be saved in either FDF or XFDF format in a separate file which can subsequently be used in the next step within your workflow. This approach minimizes the file size to just the amount needed to store your data, thus decreasing network traffic and processing time for the next step in the workflow.

For more information, see the Acrobat JavaScript Scripting Guide.

Saving Data to a Microsoft Excel Spreadsheet

Once you’ve collected PDF form data in FDF or XML format from multiple users, you can organize the form data into a comma-delimited spreadsheet file (CSV). After exporting the form data to a CSV file, you can work with the data in a spreadsheet application, such as Microsoft Excel.

You can also save form data as a tab-delimited file. Tab-delimited files can also be imported using doc object methods.

Submitting Data from Attachments

Data from various attachments can be imported into an XML or XFA-based document, and submitted to a server for processing.

Exporting Data as Images

Since images are their own fields, they can be exported in any of the standard image types using the doc object’s exportDataObject method. If the image is to be used again, it can be imported as an icon in any of the standard formats: BMP, GIF, JPEG, PCX, PNG, or TIFF.
XML Form Templates

Acrobat JavaScript defines a **Template** object that supports interactive form architectures. In this context, a **Template** is a named page within a PDF document that provides a convenient format to automatically generate and manipulate a large number of form fields. These pages contain visibility settings, and can be used to spawn new pages containing identical sets of form controls to those defined within the template.

For more information, see the *Acrobat JavaScript Scripting Guide*. 
Rights-enabled PDF Documents

When creating a PDF document, it is possible to assign it special rights that enable users of Adobe Reader to fill in forms, participate in online reviews, and attach files. Adobe LiveCycle Reader Extensions may be used to activate additional functionality within Adobe Reader for a particular document, thereby enabling the user to save, fill in, annotate, sign, certify, authenticate, and submit the document. This streamlines collaboration for document reviews and the automating of data capture with electronic forms. Users of Acrobat Professional can also Reader-enable collaboration.

This chapter consists of the following sections:

- Additional Usage Rights
- Differences between Usage Rights and Reader Enabling
- Creating Rights-enabled Documents
- Accessing Additional Usage Rights
- Applying and Validating Additional Usage Rights
- Effects of Reader Extensions on Acrobat Standard and Acrobat Professional
- Applying and Validating Additional Usage Rights

Additional Usage Rights

Additional usage rights allow an Adobe Reader user to perform specific operations on a rights-enabled PDF document. The following additional usage rights can be enabled for a PDF document (see “Creating Rights-enabled Documents” on page 89):

<table>
<thead>
<tr>
<th>Form</th>
<th>fill-in and document full-save</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>import an export</td>
</tr>
<tr>
<td>Form</td>
<td>add and delete</td>
</tr>
<tr>
<td>Form</td>
<td>submit standalone</td>
</tr>
<tr>
<td>Form</td>
<td>spawn template</td>
</tr>
<tr>
<td>Signature</td>
<td>modify</td>
</tr>
<tr>
<td>Annot</td>
<td>create, delete, modify, and copy</td>
</tr>
<tr>
<td>Annot</td>
<td>import and export</td>
</tr>
</tbody>
</table>
Permissions enabled by usage rights may be disallowed if:

- the document is encrypted and the security handler is set to disallow any of these operations, or
- the document contains DocMDP settings that do not allow these operations.

For more information, see the Acrobat JavaScript Scripting Guide.

---

**Differences between Usage Rights and Reader Enabling**

You may not develop an Adobe Reader plug-in without approval from Adobe. You must use Adobe's web-based application to describe your plug-in and submit information to Adobe. Adobe reviews it and either approves or rejects the application. The application and the Adobe Reader Integration Key Licensing Agreement can only be submitted as a web form. Information can be found at: [http://partners.adobe.com/asn](http://partners.adobe.com/asn)

Note that there is a fee to get the enabling key, and that your plug-in can only use a restricted set of APIs. As with all plug-ins, reader-enabled plug-ins must be installed on each user's machine (either through an installer or by the user), and the plug-in is then available to all PDF documents used on that computer.

PDF documents with additional usage rights contain rights-enabling information within each individual document, so no additional installation is needed. The APIs available to both plug-in and Acrobat JavaScript developers depends on which usage rights are set for each individual document; you cannot make APIs available to all documents as with Reader-enabled plug-ins. Using APIs available with additional usage rights requires no prior approval by Adobe.
Creating Rights-enabled Documents

PDF documents can be “rights-enabled” using Adobe LiveCycle Reader Extensions. Developers cannot programmatically rights-enable a PDF file.

During the design process, a PDF document may be created through the usage of Adobe LiveCycle Designer, Adobe LiveCycle Forms, or Adobe Document Server. The document creator can assign appropriate usage rights using Adobe LiveCycle Reader Extensions. The PDF document is made available on the Web, and users can complete the form on the web site, or save it locally and subsequently complete and annotate it, digitally sign it, add attachments, and return it.

In effect, Adobe LiveCycle Reader Extensions enables additional functionality within Adobe Reader for a given document. After the user has finished working with the document, those functions are disabled until the user receives another rights-enabled PDF file.

One major advantage of Adobe LiveCycle Reader Extensions is that it supports data automation through XML-based representation and transfer via SOAP, ensuring seamless integration with business logic and advanced data transport capabilities available within enterprise applications.

For more information, see the Acrobat JavaScript Scripting Guide.

Accessing Additional Usage Rights

When a rights-enabled PDF is opened, Adobe Reader changes the user interface and enables additional features. Previously unavailable APIs, for both JavaScript and plug-ins, are then available to developers.

To check whether a PDF document has rights enabled, plug-in developers can use the PDDocPermRequest methods.

JavaScript developers cannot programmatically access or assign additional usage rights with a PDF file. For this reason, it is advised that try-catch blocks be used around methods which might violate the constraints related to the document’s rights.
Applying and Validating Additional Usage Rights

The `doc` object provides two methods that rely upon its security model to apply and validate usage rights within a document:

- **appRightsSign**: Creates and embeds permission information for additional features in Adobe Reader, including a signature used to validate the permissions granted by Adobe Systems.
- **appRightsValidate**: Returns the validity of the application rights signature set by `appRightsSign`.

The usage rights supported are listed in “Additional Usage Rights” on page 87.

For more information, see the *Acrobat JavaScript Scripting Guide* and the *Acrobat JavaScript Scripting Reference*.

Effects of Reader Extensions on Acrobat Standard and Acrobat Professional

Certain editing features normally available within Acrobat Standard and Acrobat Professional are disabled for documents having Reader Extensions. This ensures that the user cannot inadvertently invalidate the additional usage rights in a document under managed review before passing the document on to an Adobe Reader user for further commenting.

For complete details, see the *Acrobat JavaScript Scripting Guide* and the *Acrobat SDK Plug-in Guide*.

Enabling Collaboration

By using Really Simple Syndication (RSS), collaboration servers can provide customized XML-based user interfaces directly inside of Acrobat itself. This provides a more dynamic and personalized tool, as well as enabling JavaScript developers to extend collaboration, including full user interfaces.

In addition, you can now migrate comments from one document to another, carry comments across multiple versions of a document, and anchor comments to content so that the annotations remain in the right place even if the content changes.

This makes it possible to automate discovery of collaboration servers, initiation workflows, and RSS feeds that may be used to populate content inside Adobe Reader.

Note that users of Adobe Acrobat Professional can enable collaboration; this enables them to invite users of Adobe Reader to participate in the review process.

For more information, see the *Acrobat JavaScript Scripting Guide* and the *Acrobat JavaScript Scripting Reference*. 
Acrobat security is similar to home security. Just as you lock your doors to prevent someone from entering your house without permission, you use Acrobat security features to lock a PDF document. For example, you can use passwords to restrict users from opening, printing, or editing PDF documents. You can use digital signatures to certify PDF documents, and you can encrypt PDF documents so that only an approved list of users can open them. To save security settings for later use, you can create a security policy that stores security settings.

**NOTE:** This chapter focuses on how to use the Acrobat 7.0.5 SDK for security. For information on how to view or modify security using Acrobat 7.0 Professional or Acrobat 7.0 Standard, see the online help for each product.

This chapter consists of the following sections:
- Acrobat Security Overview
- Encryption and Decryption
- Digital IDs and Certificates
- Security Handlers
- Digitally Signing Adobe PDF Documents
- Security Policies
- Using eEnvelopes to Send Secure Files
- Document Rights

### Acrobat Security Overview

You can use the following methods to enhance document security:

<table>
<thead>
<tr>
<th>Security Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password security</td>
<td>Add passwords and set security options to restrict opening, editing, and printing PDF documents. Acrobat supports 128-bit RC4 and 128-bit AES (Advanced Encryption Standard) security methods. You can choose which method to use when securing documents.</td>
</tr>
<tr>
<td>Certification security</td>
<td>Encrypt a document so that only a specified set of users have access to it. See “Encryption and Decryption” on page 92.</td>
</tr>
</tbody>
</table>
Encryption and Decryption

Encryption is controlled by an *encryption dictionary* in the PDF file. Acrobat uses RC4 (a proprietary algorithm provided by RSA Data Security, Inc.) to encrypt document data, and a standard (proprietary) method to encrypt, decrypt, and verify user passwords to determine whether or not a user is authorized to open a document. See Section 3.5, “Encryption,” in the *PDF Reference* for more information on the encryption used in PDF files.

Each stream or string object in a PDF file is individually encrypted. This level of encryption improves performance because objects can be individually decrypted as needed rather than decrypting an entire file. All objects, except for the encryption dictionary (which contains the security handler's private data), are encrypted using the RC4 algorithm that Adobe licenses from RSA Data Security, Inc. Plug-ins may not substitute another encryption scheme for RC4.

For digital signatures and document encryption, Acrobat supports public-key cryptography. Public-key cryptography uses two keys: a public key, which is stored inside a certificate that can be shared with other users, and a private key, called a digital ID, which is not shared with others. The public key certificate is used to encrypt (scramble) documents or to verify digital signatures, and the digital ID is used to decrypt (unscramble) encrypted documents or to create digital signatures.

Digital IDs and Certificates

A digital ID lets you create a digital signature or decrypt an encrypted PDF document. A digital ID is sometimes referred to as a private key, a credential, or a profile.

When you sign or decrypt a document, you select which digital ID to use. A digital ID is usually password-protected and can be stored on your computer as a PKCS#12 file, on a smart card, or in the Windows Credential Store. You can get a digital ID from a third-party provider, or you can create a self-signed digital ID and share your signature information with others using a public key certificate. A certificate is a confirmation of your digital ID and contains information used to protect data.

When a digital signature is applied, a unique fingerprint with encrypted numbers is embedded in the document. The recipient needs the signer's certificate to validate that the
digital signature and certificate match the signer’s digital ID. Adobe Acrobat includes one handler that has access to trusted certificates in a number of different locations. The locations include Microsoft’s cryptographic store used for Windows security, PKCS#12 encryption, which is a standard encryption format, and PKCS#11 encryption, which is used on smart cards.

Obtaining a Digital ID from a Third Party

In general, digital IDs are issued by a third party for use in any official capacity. Third-party providers, such as Entrust, include advanced security features. The provider of digital ID certificates is sometimes called a certificate authority or a signature handler. Third-party providers verify your identity, issue the private key, protect the public key, and maintain system integrity. You may want to obtain more than one digital ID if you sign documents in different roles or with different certification methods. See http://partners.adobe.com for information on using providers with advanced security features.

Digital IDs and the Acrobat SDK

You can work with digital IDs and certificates from Acrobat JavaScript and from a plug-in using the PubSec API (see “Security Handlers and Plug-ins” on page 94). Using the Acrobat SDK, you can:

● Create a digital ID (Default Certificate Security).
● Save a certificate file to disk.
● Use an existing certificate to create a digital ID.
● Manage digital IDs (Windows Certificate Security).
● Share digital ID certificates.
● Build a list of trusted identities.
Security Handlers

The code that performs user authorization and sets permissions is known as a security handler. The core API has one built-in security handler. This security handler supports two passwords:

- A user password that allows a user to open and read a protected document with whatever permissions the owner chooses
- An owner password that allows a document’s owner to change the permissions granted to users

You can either use the built-in security handler, or write a plug-in to create your own security handlers to perform user authorization in other ways (for example, by the presence of a specific hardware key or file, or by reading a magnetic card reader). You cannot create a custom security handler using Acrobat JavaScript. A custom security handler can use Acrobat or Adobe Reader’s built-in dialog boxes for entering passwords and for changing permissions.

Security handlers are responsible for:

- Setting permissions on a file.
- Authorizing access to a file.
- Setting up a file’s encryption and decryption keys.
- Maintaining the encryption dictionary of the PDF file containing the document.

Security handlers are used when:

- A document is opened — the security handler must determine whether a user is authorized to open the file and set up the decryption key that is used to decrypt the PDF file.
- A document is saved — the security handler must set up the encryption key and write any necessary security-related information into the PDF file’s encryption dictionary.
- A user tries to change a document’s security settings — the security handler must determine whether the user is permitted to perform the operation.

PPKLite and PPKMS Security Handlers

In Acrobat 6, the built-in PPKLite signature handler only supported a single certificate chain for validation; broader support required the Microsoft’s Crypto API (MSCAPI) support of the PPKMS handler. In Acrobat 7, PPKLite supports RFC3280-based certificate chains natively. MSCAPI support has been rolled into PPKLite and is no longer a separate handler.

Security Handlers and Plug-ins

The PubSec layer, introduced in Acrobat 6.0, is an interface for Acrobat public-key security handlers. PubSec forms a high-level interface to the digital signature facility; the PubSec
Providing Document Security

Security Handlers

code uses the DigSig API for digital signature operations, but provides many additional benefits. Developers are encouraged to use the PubSec HFT rather then DigSig HFT.

PubSec provides functions to:

- Count and close encrypted documents.
- Validate a specific signature field.
- Access and create digests for data buffers.
- Import and export certificate data, and manage the certificates in the Acrobat Address Book (AAB).
- Manage signature appearances (DSAP files).
- Register and un register handlers. Handlers can register as PubSec handlers to provide the following cryptographic services:
  - Perform private-key signing and signature validation
  - Act as a cryptographic source for decrypting using private keys
  - Act as a directory source for certificate-based identity authentication

Handlers can call back into the PubSec HFT for various services. Most calls to PubSec pass an opaque state object called a PSEngine. You specify a default engine upon registering the handler, and the default engine can make use of the security user interface dialog boxes provided by PubSec and DigSig.

For more information, see the Acrobat and PDF Library API Overview and the Acrobat and PDF Library API Reference.

Security Handlers and Acrobat JavaScript

SecurityHandler objects are used to access security handler capabilities such as signatures, encryption and directories. Different security handlers will have different properties and methods. Individual SecurityHandler objects may or may not implement these properties and methods.

The JavaScript interface for Adobe.PPKLite signatures was introduced in Acrobat 5.0; additions to the JavaScript interface were introduced in Acrobat 6.0. Prior to Acrobat 6.0 there was no support in Acrobat to enable JavaScript in third party security handlers.

Note that not all security handlers are JavaScript enabled, and not all JavaScript enabled handlers are enabled for all security operations. Third party public key security handlers may support JavaScript, but only if they use the new PubSec programming interface that was introduced in Acrobat 6.0.

JavaScript enabled handlers provided by Adobe include:

- The Adobe.PPKLite security handler, supporting signature and encryption, on Windows, provides directory access through the Microsoft Active Directory Scripting Interface (ADSI).
- The Adobe.AAB security handler provides a local address book and support for directory operations.
Note that the Standard security handler, used for password encryption of documents, is not JavaScript enabled in general. In Acrobat 7.0, encryption using Standard security is possible using predefined policies.

Also, in Acrobat 7.0, the Adobe.APS handler can be used for encryption via the encryptUsingPolicy method. This handler also makes a directory available via the directory services, but as no certificates are returned from this directory, it is of limited general use.

For more information, see the Acrobat JavaScript Scripting Reference.

---

Digitally Signing Adobe PDF Documents

A digital signature, like a conventional handwritten signature, identifies the person signing a document. Unlike traditional signatures on paper, however, each digital signature stores information about the person signing a document. Signatures help prevent unwanted changes to a PDF document. For example, an author may not want a PDF document with company letterhead to be changed after it is signed.

The first signature in a document is called the author signature. When you add the first signature to a document, you have the option of certifying the document. Certifying a document lets you attest to its contents and specify the types of changes allowed for the document to remain certified. Changes to the document are detected in the Signatures tab. Subsequent signatures to the document are called ordinary signatures.

To sign a document, you must select a digital ID, which contains the signature information that you can share with other users in a certificate. You can create a self-signed digital ID, or you can obtain a digital ID from a third-party provider. Using certificates, other users can validate your signatures, and you should validate the signatures of others.

---

Signing Adobe PDF Documents

An author of a PDF document can simply add a signature to indicate approval. Alternatively, a PDF document can be signed more than once and by more than one person. For example, the author can save a PDF document containing form fields as a certified document, allowing only form fields to be filled in. When another user opens the PDF document, a message indicates whether the certification is still valid. This user can then fill out the form and sign the document when finished.

To add a signature to a PDF document using Acrobat JavaScript:

- Create a signature field
- Choose a security handler
- Invoke the field’s signatureSign method

From a plug-in, you can sign a PDF document using the PubSec and DigSig APIs.
Certifying PDF Documents

You can apply permissions to a PDF document, but some applications may not respect these permissions. In these cases, it is desirable to be able to detect unauthorized changes to the document.

The author signature for a document makes Modification, Detection, and Prevention (MDP) possible. When this type of signature is applied, it is possible to certify the document, which means that you specify information about its contents and the types of changes that are allowed in order for the document to remain certified.

You certify the document by applying an author signature to it. If necessary, you can create an author signature field using Acrobat JavaScript and then sign it as described in “Signing Adobe PDF Documents” on page 96.

Validating Signatures

When you validate a signature, you verify the signer’s identity and assess any changes made after the document was signed. For an identity to be valid, the signer’s certificate, or one of its parent certificates that was used to issue the signer’s certificate, must be in your list of trusted identities, and it must not have expired or been revoked.

Using the Acrobat SDK, you can validate a signature from a plug-in, from Acrobat JavaScript, or from an external application using IAC.

Security Policies

In Adobe Acrobat 7.0, you can save security settings as policies that you can reuse. Security policies include the type of security encryption, the permission settings, and information about who can open the document or change security settings. There are two kinds of security policies:

- A user policy is developed and applied by an individual user. If you apply the same security settings to various documents, you can save time by creating a user policy and then reapplying the user policy to documents without having to specify the security settings each time. User policies for passwords and public key certificates are stored on your local computer. If you have access to Adobe Policy Server, you can also create a user policy that is stored on a policy server and is available only to the person who creates it.

- An organizational policy is created by an Adobe LiveCycle Policy Server administrator and is stored on a policy server to be shared by a group of users. Adobe Policy Server controls access to PDF documents and auditing events as defined by the security policy. You can use Adobe LiveCycle Policy Server only if your company has purchased rights and made it available to you.
Using Adobe Policy Server

Adobe Policy Server is a web server-based security system that provides dynamic control over PDF documents. Adobe Policy Server can be configured to run with LDAP, Adobe Document Server, and other enterprise systems. Policies provided by Adobe Policy Server are stored on the server and can be refreshed from the server. You must log into Adobe Policy Server to use these server policies.

Security policies are stored on a policy server, but the PDF documents are not. However, users may be required to connect to the policy server so that they can open or continue to use PDF documents to which a security policy has been applied. For information on configuring Adobe Policy Server, see the help system that appears when you start Adobe Policy Server Web Console in your browser.

Encrypting PDF Documents Using Security Policies

Acrobat JavaScript provides objects for working with security policies. There are no C APIs for manipulating security policies from a plug-in.

For more information, see the Acrobat JavaScript Scripting Guide.

Using eEnvelopes to Send Secure Files

When adding security to a document, you can choose to encrypt only the attachments. The PDF document in which documents are embedded is called an eEnvelope. Encrypting only the document attachments is especially useful for sending secure file attachments without modifying the file attachments themselves. The eEnvelope is not otherwise encrypted or permission-restricted. When other users open the eEnvelope, they can extract the file attachments and save them to disk. The saved files are identical to the original file attachments and are no longer encrypted when saved.

To create an eEnvelope using the Acrobat SDK, invoke the Acrobat JavaScript doc object’s addRecipientListCryptFilter method, which is used to encrypt data objects.

For more information, see the Acrobat JavaScript Scripting Guide.
Document Rights

By including additional usage rights in a PDF document, you can invite Adobe Reader 7.0 users to participate in an email-based document review. Additional usage rights, such as commenting rights, are document-specific and can be added to any PDF document except forms when you initiate an email-based or browser-based review.

For more information, see Chapter 10, “Rights-enabled PDF Documents.”
A PDF document can include general information such as the document’s title, author, and creation and modification dates. Such global information about the document itself (as opposed to its content or structure) is called metadata, and is intended to assist in cataloguing and searching for documents in external databases.

A PDF document can also contain object data. Object data is information associated with a particular object in a PDF document.

This chapter includes the following sections:

- Types of Metadata
- Extensible Metadata Platform (XMP)
- The Adobe XMP Toolkit
- Searching Metadata
- Object Data

### Types of Metadata

Metadata can be stored in a PDF document in either of the following ways:

- In a document information dictionary associated with the document
- In a metadata stream associated with the document or a component of the document

Metadata properties and values are represented in the World Wide Web Consortium's Resource Definition Format (RDF), which is a standard metadata format based on XML.

The set of standard metadata items is organized into schemas, each of which represents a set of properties from a particular application or industry standard. The schemas, as well as the physical representation, are defined by Adobe’s Extensible Metadata Platform (XMP) (see “Extensible Metadata Platform (XMP)” on page 102).
Document Information Dictionary

The optional **Info** entry in the trailer of a PDF file can hold a document information dictionary containing metadata for the document, including:

- title
- author
- subject
- creation and modification dates
- keywords
- creator (the original source of the document, such as Microsoft Word)
- producer (the program that produced the PDF, such as Distiller)

For more information, see the *PDF 1.6 Reference*.

Metadata Streams

Metadata, both for an entire document and for components within a document, can be stored in PDF streams called metadata streams. The advantages of metadata streams over the document information dictionary include the following:

- PDF-based workflows often embed metadata-bearing artwork as components within larger documents. Metadata streams provide a standard way of preserving the metadata of these components for examination downstream. PD-aware applications should be able to derive a list of all metadata-bearing document components from the PDF document itself.
- PDF documents are often made available on the World Wide Web or in other environments, where many tools routinely examine, catalog, and classify documents. These tools should be able to understand the self-contained description of the document even if they do not understand PDF. Besides the usual entries common to all stream dictionaries, the metadata stream dictionary contains other additional entries. See the *PDF Reference* for complete details.

    The contents of a metadata stream is the metadata represented in Extensible Markup Language (XML). This information will be visible as plain text to tools that are not PDF-aware only if the metadata stream is both unfiltered and unencrypted.

Extensible Metadata Platform (XMP)

Adobe’s Extensible Metadata Platform (XMP) is a labeling technology that allows you to embed metadata into the file itself. With XMP, desktop applications and back-end publishing systems gain a common method for capturing, sharing, and leveraging this valuable metadata — opening the door for more efficient job processing, workflow automation, and rights management, among many other possibilities.
Accessing XMP Metadata Using Acrobat JavaScript

The `metadata` method of the Acrobat JavaScript `doc` object allows you to access the XMP metadata embedded in a PDF document. The method returns a string containing the XML text stored as metadata in a particular PDF document.

Accessing XMP Metadata From a Plug-in

You can access XMP metadata information using the Acrobat SDK. Plug-ins can get or set XMP metadata using the `PDDocGetXAPMetadataProperty` and `PDDocSetXAPMetadataProperty` APIs. Using these APIs, you can access all of the properties in a `pdDoc`'s `Info` dictionary, as well as properties that are not in the `Info` dictionary.

**NOTE:** The term “XAP” refers to an early internal code name for Adobe’s Extensible Metadata Platform (XMP). For more information on this protocol, see the Adobe XMP Specification.

You can also get or set the XMP metadata associated with a `Cos` dictionary or stream, or with a `PDEContainer`. These APIs are all included in the `PDMetaDataCalls.h` header file. See the Acrobat API and PDF Library API Reference for more information on these APIs.

Acrobat Distiller and XMP Metadata

Acrobat Distiller supports embedding of XMP metadata in PDF files in two ways.

First, it always generates document level metadata. This metadata is derived from the PostScript file and cannot be modified by the PostScript program. To change the document level metadata, you must change it in the authoring application that generates the PostScript stream.

A second way of adding metadata is to create object data. See “Object Data” on page 104 for more information.

The Adobe XMP Toolkit

The XMP Toolkit is designed to help applications handle XMP operations such as the creation and manipulation of metadata. The Toolkit makes it easier for developers to support XMP metadata, and helps to standardize how the data is represented and interchanged. The XMP Toolkit can be licensed, royalty-free, from Adobe Systems.

The XMP Toolkit features a C++ interface which uses some modern (ANSI) features, such as exceptions, STL strings, and `bool`. It uses conservative coding and interface design for maximum portability and to make it easier for developers to adopt.
NOTE: Many namespaces, keywords, and related names in this document are prefaced with the string “XAP”, which was an early internal code name for XMP metadata. These names are retained for compatibility.

The XMP Toolkit consists of two parts:

- **MetaXAP** manages the metadata for a managed resource such as an application document file. It defines the objects that act as containers for properties relating to a specific document, and is the primary interface to the XMP Toolkit. MetaXAP provides the top level abstraction for metadata about a document. Nodes are accessed via string pathnames which use a simplified form of XPath strings (XML Path Language: [http://www.w3.org/TR/xpath](http://www.w3.org/TR/xpath)).

- **UtilityXAP** provides a variety of special purpose utilities to simplify common uses of MetaXAP. For example, MetaXAP reads and writes property values as strings. UtilityXAP has services that include conversion between integers and other types.

XMP metadata properties are organized by schemas. In RDF, the schema is defined by a namespace attribute. Within each schema, properties are named via a path string. This path string has a very simple syntax which is modelled on the XPath standard.

The full XMP data model is supported, including values that are simple literals, nested descriptions, and structured containers. Applications should include only the `XAPToolkit.h` file, and optionally the `UtilityXAP.h` file.

---

### Searching Metadata

You can search metadata both by using Acrobat JavaScript and from a plug-in.

In Acrobat JavaScript, use the `docInfo` and `docXMP` properties of the `Search` object.

In a plug-in, search metadata by setting the `kSearchDocInfo`, `kSearchDocumentXMP`, or `kSearchEveryWhere` search option flags.

---

### Object Data

The PDF structure tree and marked content work in tandem to support object data. Object data is information associated with a particular object in a PDF document, such as a chair in an engineering drawing.
Creating Object Data

Object data is generated by the authoring application, such as Microsoft Visio. You can also attach additional name/value pairs to a set of content objects.

You can generate object data by inserting pdfmarks into the PostScript file to be processed by Distiller. Using pdfmarks requires that you know PostScript well enough to either modify the PostScript file directly, or modify the authoring application to generate the appropriate pdfmarks in the PostScript stream. For more information, see “Choosing an Adobe PDF Settings File” on page 41, the pdfmark Reference Manual, and the PDF Reference.

In an existing PDF document, you can use the PDSEdit APIs to create object data programatically from a plug-in.

Searching and Accessing Object Data

Beginning in Acrobat 7.0, you can access object data from a plug-in and from Acrobat JavaScript.

From a plug-in, you use C APIs to:

● Enumerate the PDSElement objects in a document which have user properties associated with them

● Enumerate the object data associated with a PDSElement object as either ASText or CosObj objects

● Check whether a PDF document or specific PDSElement has any user properties associated with it

● Find the ancestor or child node in the structure tree of a PDSElement that has user properties

● Search for object data using the kSearchUserProperties flag to the Search plug-in.

You can search object data from Acrobat JavaScript by using the Search object’s objectmetadata property. There are no C APIs to search for object-level metadata from a plug-in.
Working with Metadata

Object Data
Using Acrobat and the Acrobat SDK, you can:

- **Search** - locate an occurrence of a word in the document content, metadata, attachments or other objects.
- **Catalog** - create a full-text index of your PDF document or document collections.

This chapter consists of the following sections:

- The Acrobat Search Plug-in
- Indexing PDF Documents

### The Acrobat Search Plug-in

With Acrobat, Adobe provides a full-text search system. Using the Acrobat search system, you can:

- **Initiate a search with options**
  
  You can pass query expressions and search settings (for example, whole words only, word stemming, case sensitive, and so forth) to the Acrobat search plug-in and initiate the search. Search results will be presented to the user.

- **Control the list of indexes to search**
  
  The search interface allows searches to be performed on one or more of the available indexes. You can control the list of active indexes.

The Acrobat Search plug-in allows users to perform text searches in PDF documents. It adds menus, menu items, toolbar buttons, and a Search panel to the Acrobat or Adobe Reader. You can communicate with the Search plug-in via its plug-in API, via IAC (DDE or Apple events) or via Acrobat JavaScript’s `Search` object. You can perform all search options from each of these development environments.

### Creating Custom Searches

The Acrobat interface can be extended with new menus and toolbar icons. For example, buttons to invoke a search and to find the next or previous occurrences can be added to the toolbar. You can also add custom, repeatable searches to a menu to increase user efficiency.

For more information about adding menu items and toolbar buttons, see “Adding or Removing Menus and Menu Items” on page 59 and “Modifying the Toolbar” on page 60.
Search Options

Using the Acrobat JavaScript `Search` object's properties and using `SearchWordOptions` flags with the Search API methods from a plug-in, you can specify the scope and search options for your search.

You can set the scope of the search to be one or more of the following:

- Document text
- Bookmarks
- Document metadata (XMP)
- Signatures
- DocInfo fields
- Object data
- Attachments
- EXIF data in associated JPEG images

Matching Criteria

You can also specify whether the search should match:

- an exact phrase
- all words without regard to order
- any of the words

You can also specify whether case and diacritical marks are used or ignored in the search.

Searching across Multiple Documents

Using the Acrobat SDK, you can search any of the following:

- the active document
- a specified folder
- a specified index
- all active indices

Replacing the Acrobat Search Plug-in

The Acrobat Search plug-in provided with Acrobat is a true Acrobat plug-in. You can remove it from the system by simply removing it from the Plug-ins directory and restarting Acrobat.

Full-text search developers can create a custom search plug-in using the Acrobat core API. This search plug-in can be represented as one or more toolbar buttons or menu items, and it can use the capabilities provided in the API.
When creating a replacement search plug-in for Acrobat, you must decide what indexes your search plug-in will use. You can either create your own indexes (see “Extracting and Highlighting Text”) or search the Lextek indexes created by the Acrobat 7 Catalog plug-in.

Indexing PDF Documents

You can use the Acrobat SDK to create a full-text index of a set of PDF documents. A full-text index is a searchable database of all the text in the documents. After building an index, you can use search the entire library quickly.

You can build and manipulate indices from a plug-in, from Acrobat JavaScript or from an external application using IAC (DDE or Apple events) calls.

Extracting and Highlighting Text

For indexing PDF files, Acrobat provides text extraction APIs. Text extraction also supplies position information that can be used to highlight search hits in the original PDF file. The text extraction tools are provided as calls in the plug-in API on the Acrobat platforms (Mac OS and Windows).

You can extract ASCII text from a PDF file using a plug-in or using Acrobat JavaScript. You can also save the PDF document as text or rich text.

Indexing and Acrobat JavaScript

It is possible to extend and customize indexes for multiple PDF documents using the Acrobat JavaScript Catalog, CatalogJob, and Index objects. These objects may be used to build, retrieve, or remove indexes.

The Index object represents a Catalog-generated index and contains a build method that is used to create an index.

For more information, see the Acrobat JavaScript Scripting Guide.

The Acrobat Catalog Plug-in

Acrobat Catalog is a plug-in that allows you to create a full-text index of a set of PDF documents. The Catalog plug-in has an HFT consisting of several methods that plug-in developers can import and use. In addition, Catalog supports DDE, and broadcasts several Windows messages.

For more information, see the Acrobat and PDF Library API Overview.
Accessibility

Acrobat and the Acrobat SDK provide extensive support for accessibility of documents to visually-impaired users.

This chapter consists of the following sections:

● Accessibility Overview
● Optimizing the Accessibility of PDF Documents
● Making Forms Accessible
● Annotations and Accessibility
● Accessibility and Security
● Microsoft’s Active Accessibility (MSAA) Interface

Accessibility Overview

To enable proper vocalization, either through a screen reader or by some more direct invocation of a text-to-speech engine, PDF supports the following features:

● Specifying the natural language used for text in a PDF document—for example, as English or Spanish
● Providing textual descriptions for images or other items that do not translate naturally into text, or replacement text for content that does translate into text but is represented in a nonstandard way (such as with a ligature or illuminated character)
● Specifying the expansion of abbreviations or acronyms

The core of this support lies in the ability to determine the logical order of content in a PDF document, independently of the content’s appearance or layout, through logical structure and tagged PDF documents (see “Creating Tagged PDF Documents” on page 43). Tagged PDF documents are created using the Acrobat Professional user interface or using the Acrobat SDK.

An accessibility application can extract the content of a document for presentation to a disabled user by traversing the structure hierarchy and presenting the contents of each node. For this reason, producers of PDF files must ensure that all information in a document is reachable via the structure hierarchy, and they are strongly encouraged to use the facilities described in this section.


For complete details on accessibility support in PDF files, see the PDF Reference.
Optimizing the Accessibility of PDF Documents

If a PDF document is untagged, you can convert it to a tagged PDF using Acrobat Professional.

**NOTE:** Acrobat Standard provides only minimal support tagging and no way to review or repair accessibility problems.

On the tagged PDF document, perform an accessibility Full Check to isolate problems with PDF accessibility.

To help improve the accessibility of tagged PDF documents, you can use the Acrobat user interface or the Acrobat SDK to:

- Create document navigation by adding bookmarks to the document (see “Adding and Managing Bookmarks” on page 47)
- Add alt text to images
- Create hyperlinks, which enhances accessibility because it is easier to activate a hyperlink than to search for a specific reference point in the same or another document (see “Adding and Managing Links” on page 49)
- Add watermarks that optimize the accessibility of the PDF document (see “Manipulating Pages of a PDF File” on page 50)
- Manipulate the tag structure tree using the PDSEdit APIs from a plug-in (see the Acrobat and PDF Library API Overview)

Making Forms Accessible

The following accessibility tips can make your forms more easily usable by all users.

**Field Descriptions**

Every field that is not hidden should contain a user-friendly name. This name is spoken when a user acquires the focus to that field and should give an indication of the field’s purpose. For example, if a field is named `name.first`, a good short description would be **FirstName**. The name should not depend on the surrounding context. For instance, if a form asks for the first name of both a person and their spouse, the field in the spouse section might be named **Spouse's FirstName**.

This description is also displayed as a tooltip when the user positions the mouse over the field.
Setting Tab Order

Because most users use the tab key to traverse a form, the tab order of the fields should be set in a logical way. This is especially important for visually impaired users, as they cannot rely on mouse movements or visual clues.

Pressing the tab (shift-tab) key when there is no form field that has the focus causes the first (last) field in the tab order on the current page to become active. If there are no form fields on the page, Acrobat alerts the user of this via a speech cue.

Annotations and Accessibility

In Acrobat 6.0, annotations in PDF documents were not fully and easily accessible. This presented a challenge to visually-impaired users who access Acrobat through assistive technologies, such as screen readers and screen magnifiers.

The Acrobat 7.0.5 SDK adds a new annotation accessibility API for the generalized interface between annotations and accessibility so that new and custom annotation types can be supported more easily and reliably.

A new callback function, `AVAnnotHandlerGetAccessibilityString`, has been added to the `AVAnnotHandler` record. Note that there is one annotation handler for each annotation type. For more information, see the Acrobat and PDF Library API Reference.

Tagged Annotations

Annotations can be dynamically associated with a new structure tree that is separate from the original content of the document. This supports accessibility without modifying the original content.

Text-To-Speech

Text-To-Speech engines can translate structured text in a PDF document into audible sound.

To access the Text-To-Speech engine from Acrobat JavaScript, use the `TTS` object, which has methods to render text as digital audio and present it in spoken form to the user.

Accessibility and Security

It is not necessary to sacrifice security in order to make a document accessible: select the Enable Text Access for Screen Reader Devices for the Visually Impaired option in the Security section of Document Properties.
You can enable this option using Acrobat JavaScript's `doc` object's `encryptForRecipients` method. This method allows you to control whether content can be extracted for use by applications that, for example, read text aloud.

For more information, see the Chapter 11, “Providing Document Security”.

---

**Microsoft’s Active Accessibility (MSAA) Interface**

Acrobat exports COM objects in order to make the content of a PDF file available to assistive technology. An accessibility application such as a screen reader can interface with Acrobat through Microsoft’s Active Accessibility (MSAA) interface, using MSAA objects that Acrobat exports, or directly through exported COM objects that allow access to the PDF’s internal structure, called the document object model, or DOM.

Information about the underlying PDF structure is made available through direct COM objects that represent the PDF DOM.

The DOM accessibility interfaces provide more extensive access. The DOM and MSAA models are related, and a developer can use either or both. Acrobat issues notifications to accessibility clients about significant events occurring in the PDF file window, and responds to requests from such clients.

For more information, see *Reading PDF Through COM Accessibility Interfaces*. 
Adobe PDF layers are components of content that may occupy the same physical space as other components. Multiple components may be visible or invisible depending on their settings, and may be used to support the display, navigation, and printing of layered PDF content by various applications.

Using the Acrobat SDK, you can display, navigate and print layered PDF documents.

This chapter consists of the following sections:
- About Adobe PDF layers
- Creating Layered PDF Files
- Navigating with Layers
- Working with Layers from Acrobat JavaScript
- Working with Layers from a Plug-in

### About Adobe PDF layers

Acrobat supports the display, navigation, and printing of layered Adobe PDF content output by applications such as Adobe InDesign, AutoCAD, and Microsoft Visio.

PDF layers are supported through the usage of PDF Optional Content Group (OCG) objects. Optional content refers to content in a PDF document that can be selectively viewed or hidden. Optional content became available in Acrobat 6.

Note the following:
- You can rename and merge layers, change the properties of layers, and add actions to layers. You can also lock layers to prevent them from being hidden.
- You can control the display of layers using the default and initial state settings. For example, if your document contains a copyright notice, you can easily hide the layer whenever the document is displayed onscreen but ensure that the layer always prints.
- Acrobat does not allow you to author layers that change visibility according to the zoom level, but it does support this capability from other authoring applications.
- To direct users to a custom view using a particular layer set, you can add bookmarks to a PDF document that contains layers. Use this technique to highlight a portion of a layer that is especially important. You can add links so that users can click on a visible or invisible link to navigate to or zoom in on a layer.
- To create layers while exporting InDesign CS or later documents to PDF, make sure that Compatibility is set to Acrobat 6 (PDF 1.5) and that Create Acrobat Layers is selected in the Export PDF dialog box.
Creating Layered PDF Files

The PDFMakers for several engineering applications, including Microsoft Visio and AutoCad, automatically generate the necessary ProcSets to create layered PDF documents. Third-party developers must insert OC ProcSet information into the PostScript stream. For more details, see “Creating PDF Documents from an Authoring Application” on page 39 and the Acrobat Distiller API Reference.

You create layered PDF documents with the ProcSet used to build Optional Content (OC) into PDF through Acrobat Distiller.

Navigating with Layers

Since information can be stored in different layers of a PDF document, navigational controls can be customized within different layers, whose visibility settings may be dynamically customized so that they are tied to context and user interaction. For example, if the user selects a given option, a set of navigational links belonging to a corresponding optional content group can be shown.

Working with Layers from Acrobat JavaScript

Using Acrobat JavaScript, you can determine the order in which layers are displayed in the user interface.

You can also use Acrobat JavaScript to:

● Merge layers in a PDF document
● Flatten layers in a PDF document
● Combine PDF layered documents

For more information, see the Acrobat JavaScript Scripting Guide.
Working with Layers from a Plug-in

The PDOCG object represents an optional-content group. This corresponds to a PDF OCG dictionary representing a collection of graphic objects that can be made visible or invisible. Any graphic content of the PDF can be made optional, including page contents, XObjects, and annotations. The specific content objects in the group have an OC entry in the PDF.

Using PDOCG methods from a plug-in, you can:

- Create a new OCG
- Get and set the current state of an OCG
- Get and set the initial OCG state
- Get and set document configurations

For more information, see the Acrobat and PDF Library API Overview and the Acrobat and PDF Library API Reference.
Index

A
"about" box and splash screen 64
accessibility 111
  forms 112
  security 113
  tagged PDF 111
Acrobat Capture 3.0.x 46
Acrobat Distiller 39
  batch processing 42
  parameters 42
  settings file 41
  XMP 103
acrobat forms 80
Acrobat JavaScript 19
  contexts 20
  cross-platform compatibility 32
  distribution 32
  forms 82
  JObject 27
  objects 19
  samples 35
  scope 32
Acrobat SDK
  Unix 19
AcroForm 83
AcroPDF 26
actions 51
ADBC 56
Adobe Document Server 46, 89
Adobe LiveCycle Designer 79, 84, 89
Adobe LiveCycle Document Security 16
Adobe LiveCycle Forms 89
Adobe LiveCycle Reader Extensions 16, 87, 89
Adobe Policy Server 16, 92, 98
Adobe Reader
  enabling 88
Apple events
  Distiller 40
  appModel 84
AS layer 24
  attachments
    searching 108
author signature 97
Autodesk AutoCAD 37
AV layer 23

B
barcode 84
bookmark 49
bookmarks
  searching 108

C
Catalog 109
Catalog Plug-in 24, 109
CatalogJob 109
certificates 92
certified plug-in 21
collaboration 90
contexts 20
core API
  organization 22
core API organization
  AS layer 24
  AV layer 23
  Cos layer 24
  PD layer 23
  PDFEdit 23
  PDSEdit 23
  platform-specific methods 24
Cos layer 24
cross-platform compatibility 32

D
DDE 27
  PDF creation 39
Index

dictionary 55
digital ID 92
digital signature 96
Digital Signature Plug-in 24
Distiller 39
    Apple events 40
Distiller
    OLE Automation 40
distribution 32
DNS-SD 75
doc
    addRecipientListCryptFilter 98

E
embedded fonts 57
encryption 92
ExecuteThisJavaScript 83
ExecuteThisScript 31
extended APIs
    Catalog 24
digital signature 24
    forms 24
    PDF consultant 24
    SaveAsXML 25
    search 25
    spelling 25
    weblink 25

F
FDF 68, 85
footers 54
forms
    acrobat 80
    Acrobat JavaScript 82
    barcode fields 84
    FDF 85
    field types 81
guidelines 82
    Microsoft Excel 85
    OLE automation 83
plug-in 83
populating 83
templates 86
web-ready 84
workflows 81
XDP 85
XFDF 85
Forms API 83
Forms Plug-in 24

H
handlers
    security 94
Hanko stamp 71
headers 54
help files 64
Host Function Table (HFT) 24
hyperlinks 50

I
IAC 30
    DDE 27
    JSObject 27
    OLE support 26
    PDF Browser Controls 30
    PDF browser controls 26
    samples 35
Index 109
Inkan stamp 71
installation 32

J
JSObject 22, 27, 31

L
Links 50
logical structure 53

M
Mac OS X 22
Mach-O 22
<table>
<thead>
<tr>
<th>MDP</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>menus and menu items</td>
<td>59</td>
</tr>
<tr>
<td>metadata</td>
<td>108</td>
</tr>
<tr>
<td>Microsoft Access</td>
<td>37</td>
</tr>
<tr>
<td>Microsoft Active Accessibility (MSAA)</td>
<td>114</td>
</tr>
<tr>
<td>Microsoft Excel</td>
<td>37, 85</td>
</tr>
<tr>
<td>Microsoft Internet Explorer</td>
<td>37</td>
</tr>
<tr>
<td>Microsoft Outlook</td>
<td>37</td>
</tr>
<tr>
<td>Microsoft PowerPoint</td>
<td>37</td>
</tr>
<tr>
<td>Microsoft Project</td>
<td>37</td>
</tr>
<tr>
<td>Microsoft Publisher</td>
<td>37</td>
</tr>
<tr>
<td>Microsoft Visio</td>
<td>37</td>
</tr>
<tr>
<td>Microsoft Word</td>
<td>37</td>
</tr>
<tr>
<td>MSCAPI</td>
<td>94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>O</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OBDC</td>
<td>56</td>
</tr>
<tr>
<td>object data</td>
<td>108</td>
</tr>
<tr>
<td>OCG</td>
<td>54</td>
</tr>
<tr>
<td>OLE Automation</td>
<td>40</td>
</tr>
<tr>
<td>Distiller API</td>
<td>40</td>
</tr>
<tr>
<td>OLE support</td>
<td>26</td>
</tr>
<tr>
<td>online collaboration</td>
<td>67</td>
</tr>
<tr>
<td>organization of core API</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>page content</td>
<td>52</td>
</tr>
<tr>
<td>pages</td>
<td>52</td>
</tr>
<tr>
<td>creating empty</td>
<td>51</td>
</tr>
<tr>
<td>deleting</td>
<td>51</td>
</tr>
<tr>
<td>inserting</td>
<td>51</td>
</tr>
<tr>
<td>moving</td>
<td>51</td>
</tr>
<tr>
<td>PD layer</td>
<td>23</td>
</tr>
<tr>
<td>PDDocCreate</td>
<td>45</td>
</tr>
<tr>
<td>PDDocPermRequest</td>
<td>89</td>
</tr>
<tr>
<td>PDF Browser Controls</td>
<td>30</td>
</tr>
<tr>
<td>PDF Consultant Plug-in</td>
<td>24</td>
</tr>
<tr>
<td>PDF documents</td>
<td>37</td>
</tr>
<tr>
<td>creating</td>
<td>37</td>
</tr>
<tr>
<td>printing</td>
<td>56</td>
</tr>
<tr>
<td>tagged</td>
<td>43</td>
</tr>
<tr>
<td>PDF printer</td>
<td>38</td>
</tr>
<tr>
<td>PDFEdit</td>
<td>23, 52</td>
</tr>
<tr>
<td>samples</td>
<td>52</td>
</tr>
<tr>
<td>PDFField</td>
<td>83</td>
</tr>
<tr>
<td>PDFMaker</td>
<td>37</td>
</tr>
<tr>
<td>pdfmark</td>
<td>39, 41, 44</td>
</tr>
<tr>
<td>pdfmarks</td>
<td>53</td>
</tr>
<tr>
<td>PDESedit</td>
<td>23, 53, 112</td>
</tr>
<tr>
<td>Plug-in</td>
<td>19</td>
</tr>
<tr>
<td>plug-ins</td>
<td>19</td>
</tr>
<tr>
<td>Adobe Reader-enabled</td>
<td>21</td>
</tr>
<tr>
<td>certified</td>
<td>21</td>
</tr>
<tr>
<td>cross-platform compatibility</td>
<td>32</td>
</tr>
<tr>
<td>installation</td>
<td>32</td>
</tr>
<tr>
<td>scope</td>
<td>32</td>
</tr>
<tr>
<td>PostScript</td>
<td>39</td>
</tr>
<tr>
<td>PPKLite</td>
<td>94</td>
</tr>
<tr>
<td>PPKMS</td>
<td>94</td>
</tr>
<tr>
<td>printing</td>
<td>56</td>
</tr>
<tr>
<td>PubSec layer</td>
<td>94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader enabling</td>
<td>88</td>
</tr>
<tr>
<td>Reader extensions</td>
<td>88</td>
</tr>
<tr>
<td>effects</td>
<td>90</td>
</tr>
<tr>
<td>Reader-enabled plug-ins</td>
<td>21</td>
</tr>
<tr>
<td>Really Simple Syndication (RSS)</td>
<td>90</td>
</tr>
<tr>
<td>rights-enabling</td>
<td>87</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>samples</td>
<td>34</td>
</tr>
<tr>
<td>IAC</td>
<td>35</td>
</tr>
<tr>
<td>JavaScript</td>
<td>35</td>
</tr>
<tr>
<td>SaveAsXML</td>
<td>24</td>
</tr>
<tr>
<td>XML conversion</td>
<td>76</td>
</tr>
<tr>
<td>SaveAsXML plugin</td>
<td>25</td>
</tr>
<tr>
<td>scope</td>
<td>32</td>
</tr>
<tr>
<td>Search Plug-in</td>
<td>25</td>
</tr>
<tr>
<td>Search plug-in</td>
<td>107</td>
</tr>
<tr>
<td>searching</td>
<td></td>
</tr>
</tbody>
</table>
Index

attachments 108
  custom 107
metadata 108
object data 108
XMP 108
security 91
  accessibility 113
security handlers 94
Security Policies 97
snippets 52
SOAP 74, 84
Spelling 55
Spelling Plug-in 25
splash screen 64
stamps 71

tagged PDF 53, 111
tagged PDF documents 43
template 74
templates 86
text extraction 109
text-to-speech (TTS) engines 113
thumbnails 50
toolbar 49

Unix 19
usage rights 21, 87
effects 90
user interface
  "about" box and splash screen 64
  help files 64
  menus and menu items 59
toolbar 49

Visual Studio . NET 2003 22

W
watermarks 54
Weblink Plug-in 25
workflows
  forms 81

X
XDP 85
XFDF 85
XML 73
  architecture 73
  templates 74
XML Forms
  forms
  XML 74
XMP 74
  searching 108
XMP Toolkit 103
XSD 75