



Guide to SDK Samples

June 2007

Adobe® Acrobat® SDK

Version 8.1

© 2007 Adobe Systems Incorporated. All rights reserved.

Adobe® Acrobat® SDK 8.0 Guide to SDK Samples for Microsoft® Windows®, Mac OS®, Linux®, and UNIX®

Edition 2.0, June 2007

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names and company logos in sample material are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, Flex, PostScript and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apple and Mac OS are trademarks of Apple Computer, Inc., registered in the United States and other countries.

JavaScript is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Microsoft and Windows are either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

All other trademarks are the property of their respective owners.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contents

Preface	6
What's in this guide?	6
Who should read this guide?	6
Related documentation	6
1 Plug-in Samples	8
ActionHandler	8
BasicPlugin.....	9
BatchCommand.....	10
CapiSamples.....	11
CivilDiscovery	13
DMSIntegration.....	14
DdeServer.....	15
DocSign.....	16
Embed3DData	17
ExternalView.....	19
ExternalWindow	20
ExtractBatesInfo	21
PDFBinder	22
ProgressMonitor	23
RplcFileSystem	24
SecurityHandler	25
SelectionServer	26
ShowPermissionsSDK7	27
Stamper.....	28
Starter	29
Transparency.....	30
UncompressPDF	31
WeblinkDemo.....	32
WordFinder.....	33
2 JavaScript Samples	34
ADMCollection	34
AddSignature.....	35
AddToolButton.....	36
AnnotSample	37
AnnotatedWords	38
ConvertTime.....	39
DeleteNoCommentPages	40
Eliza.....	41
EmbeddingFormData	42
EventState.....	43
GetStockPrice.....	44
GoToBookmark	45
JSCollection	46

2 JavaScript Samples (Continued)

JavaScriptSnippet1	47
OCGApiSample.....	48
PresentationMonitor	49
PresentationNote	50
SOAPCollabSample.....	51
ScriptEvents.....	52
SigningWorkflow.....	53
SilentPrint.....	54
TextExtract	55
Toolbarbutton	56
Trophy	57
TwoPartInvention.....	58

3 Mac OS - Interapplication Communications 59

AddAnnotations	59
DistillerControl	60
ObjectProperties.....	61
PrintPage	62
RotatePages.....	63
SelectText.....	64
WatermarkJsoAS.....	65

4 Windows - Interapplication Communications..... 66

AcroPDFInHTML.....	66
AcrobatActiveXVB.....	67
ActiveViewVB.....	68
ActiveViewVC.....	69
AdobePDFSilent.....	70
BasicIACVB	71
BasicIacCS.....	72
BasicIacJsoVB	73
BasicIacOCXCS.....	74
BasicIacVC	75
DdeOpenVC.....	76
DistillerCtrlVB	77
DistillerCtrlVC.....	78
DistillerCtrlWMVC.....	79
ExecuteScriptIacVB	80
FillFormCS	81
FormsAutomationVB.....	82
JSOFindWordVB.....	83
JSObjectAccessVB	84
RemoteControlAcrobatVC	85
SearchPdfVB	86
StaticViewVB	87
StaticViewVC	88
WatermarkJsoVB.....	89

5 Tools	90
APILocator.....	90
Plug-in Wizard	91
Reader-enabling Tools.....	92
ShowPermissions	93
VerifyURLs	94

Preface

This guide describes the samples and support available within the Adobe® Acrobat® Software Development Kit (SDK).

What's in this guide?

It is possible to extend and customize Acrobat Professional, Acrobat Standard or Adobe Reader® by using the Acrobat SDK to write JavaScript® code, implement interapplication communication, and write plug-ins. This guide provides descriptions of samples and tools meant to assist you with such development efforts.

Who should read this guide?

This guide is meant for developers who would like to extend or customize Acrobat or Adobe Reader using JavaScript, interapplication communication, or plug-ins.

For information about Acrobat SDK technologies and the many ways that developers can extend Acrobat or Adobe Reader using the Acrobat SDK, see the *Overview*.

Related documentation

The following resources and samples provide further information about the Acrobat SDK, as well as additional documents that you should have available for reference.

For information about	See
A guide to the documentation in the Acrobat SDK.	<i>Acrobat SDK Documentation Roadmap</i>
Known issues and implementation details.	<i>Readme</i>
Answers to frequently asked questions about the Acrobat SDK.	<i>Developer FAQ</i>
New features in this Acrobat SDK release.	<i>What's New</i>
A general overview of the Acrobat SDK.	<i>Overview</i>
A guide to the sections of the Acrobat SDK that pertain to Adobe Reader.	<i>Developing for Adobe Reader</i>
Prototyping code without the overhead of writing and verifying a complete plug-in or application.	<i>Snippet Runner Cookbook</i>
Configuring and administering a system for online collaboration using comment repositories, Acrobat and Adobe Reader.	<i>Acrobat Online Collaboration: Setup and Administration</i>

For information about	See
Enabling Acrobat to save documents in a customized text-based format.	<i>Extending the Acrobat SaveAsXML Plug-in</i>
Using DDE, OLE, Apple events, and AppleScript to control Acrobat and Adobe Reader and to render Adobe PDF documents.	<i>Developing Applications Using Interapplication Communication</i>
Detailed descriptions of DDE, OLE, Apple event, and AppleScript APIs for controlling Acrobat and Adobe Reader or for rendering PDF documents.	<i>Interapplication Communication API Reference</i>
Detailed descriptions of JavaScript APIs for adding interactivity to 3D annotations within PDF documents.	<i>JavaScript for Acrobat 3D Annotations API Reference</i>
Using JavaScript to develop and enhance standard workflows in Acrobat and Adobe Reader.	<i>Developing Acrobat Applications Using JavaScript</i>
Detailed descriptions of JavaScript APIs for developing and enhancing workflows in Acrobat and Adobe Reader.	<i>JavaScript for Acrobat API Reference</i>
Using RSS to track remote resources in an occasionally-connected environment.	<i>Acrobat Tracker</i>
Detailed descriptions of APIs for controlling Acrobat Distiller for PDF file creation.	<i>Acrobat Distiller API Reference</i>
Specifying settings for the creation of PDF files.	<i>Adobe PDF Creation Settings</i>
A detailed description of an extension to the PostScript® language which allows the description of PDF features not found in standard PostScript.	<i>pdfmark Reference</i>
A detailed description of the PDF file format.	<i>PDF Reference</i>
Developing plug-ins for Acrobat and Adobe Reader, as well as for PDF Library applications.	<i>Developing Plug-ins and Applications</i>
Detailed descriptions of the APIs for Acrobat and Adobe Reader plug-ins, as well as for PDF Library applications.	<i>Acrobat and PDF Library API Reference</i>
Detailed descriptions of the APIs for using assistive technology with PDF documents.	<i>PDF Accessibility API Reference</i>
Using JavaScript to perform repetitive operations on a collection of files.	<i>Batch Sequences</i>
A detailed description of the parameters for opening PDF files and for performing actions on them using a URL or command.	<i>Parameters for Opening PDF Files</i>
A list of the U3D elements supported by Acrobat.	<i>U3D Supported Elements</i>

ActionHandler

Location

Plugins/Samples/ActionHandler

Description

Demonstrates how to create a new PDAction type. PDActions can be tied to various events within the viewer, for example the mouse up over link annotations, page open events, form field triggers and so on. ActionHandler registers the Display Help Message action which allows the user to supply a text string that is displayed when the action is executed.

Usage

Within the action selection dialog box, select the Display Help Message action type in the drop-down list. The action has a single property - the string to be displayed when the action is executed. Users can configure the action (in this case, enter a text string) by clicking the Edit button.

Implementation details

An action handler consists of the series of callbacks contained in the AVActionHandlerProcsRec structure. The callbacks are used by the viewer to obtain a name for the action, to obtain the text strings that customize the action selection dialog box, for presenting an edit properties dialog box, and for creating the action dictionary in the PDF file. The routine RegisterActionHandler is called during plug-in initialization. It initializes the callback structure and then registers it using the following code:

```
AVAppRegisterActionHandler (&gActionHandler, NULL, "ADBE:ActionHandler",  
"Display Help Message");
```

The parameters are the structure, (NULL in this case since we have no user data that we want to pass to each routine) the name of this action handler, and a text string for the popup list item.

Note: The plug-in will log messages to SnippetRunner's CommonInterfaceFX.swf output pane, if the SnippetRunner is installed and the CommonInterface is running.

BasicPlugin

Location

Plugins/Samples/BasicPlugin

Description

Provides the minimum code required to enable developers to get started. The code adds a new menu item under the Acrobat SDK menu and displays a simple message. Recall that in the Acrobat viewers, the Acrobat SDK submenu is installed on the Advanced menu; in Adobe Reader it is installed on the Tools menu.

With the basic plug-in framework set up in the sample, users can quickly make a plug-in of their own by modifying and adding the code in the BasicPlugin.cpp file to set up their own menu item and menu function.

BatchCommand

Location

Plugins/Samples/BatchCommand

Description

Demonstrates how to implement an AVCommand handler. An AVCommand handler can expose functionality through the AVCommand API and/or the batch framework.

Usage

Follow these steps to test the functionality of the sample:

- Click Advanced > Batch Processing... > Edit Batch Sequences and create a new batch sequence.
- Click Select Commands and add the Isolate Form Data and Lock Form Fields commands to the sequence.
- Configure the command to process one or more PDF documents that contain Acrobat forms.

Implementation details

BatchCommand shows that it is possible for a single AVCommandHandler to implement multiple AVCommands. Where appropriate, the callbacks in the handler are passed the AVCommand that is being manipulated. This allows the handler to execute code specific to a particular type of command.

Note: Will not work in Adobe Reader due to PDF page content access. The standard batch process will work in Adobe Reader.

CapiSamples

Location

Plugins/Samples/CapiSamples

Description

There are two samples in this demonstration. One is SampleCsp. It is a DLL, and it is the actual Cryptographic Service Provider (CSP). The other one is SampleRegistrar that is to use the services provided by SampleCsp.

The purpose of SampleCsp is to provide a reference implementation which can be used with Acrobat 6+. The cryptographic functionality of this CSP is all passed through to the Microsoft Enhanced Crypto Provider. This implementation currently works on PFX files. SampleCsp opens the PFX files and uses the credential within them.

SampleRegistrar is a separate application allowing users to manage the certificates bound to SampleCsp. This application interacts with SampleCsp to do the following:

- Register a digital ID file (PFX file) and add its certificates to the windows "My" store.
- Un-register a digital ID file and remove its certificates from the windows "My" store.
- List the registered digital IDs.

Usage

To be able to run on Microsoft operating systems, a CSP must be digitally signed by Microsoft. A Microsoft signed version of SampleCsp can be located under the MsSignedSampleCsp directory. The test signing utility cspSign.exe can be downloaded from Microsoft website <http://www.microsoft.com/downloads/details.aspx?familyid=0F436C75-2304-42BB-B81A-BA0C2C47BAC2&displaylang=en>

To manage certificates by SampleRegistrar, follow these steps:

1. Compile SampleRegistrar.
2. Register SampleCsp by doing the following:
 - At Dos prompt, go to the MsSignedSampleCsp directory.
 - Type in Command 'regsvr32 SampleCsp.dll'.
3. At Dos prompt, go to the directory where SampleRegistrar.exe resides.
4. Use the following commands to manage certificates:
 - To list registered PFXs: 'SampleRegistrar.exe -l'
 - To register PFX: 'SampleRegistrar.exe -r mycert.pfx password'
 - To un-register PFX: 'SampleRegistrar.exe -u <hex-encoded-sha1-cert-digest>'

There are some easy tests you can run to check the CSP is working with Acrobat:

1. Register a digital ID via the CSP. It must show up in the Security Settings under "Windows Digital IDs".
2. Try to make a digital signature with the registered digital ID.
3. Encrypt a file for this certificate and then try to decrypt it. You need to close the document and reopen it.

If all of these tests pass, then the CSP is working. The effect should be to un-register the digital ID, not to actually delete the file.

CivilDiscovery

Location

PluginSupport/Samples/CivilDiscovery

Description

Demonstrates an automated workflow for the production phase of discovery in support of civil litigation. In this phase, certain pages that are extracted from evidentiary documents (PDF files), may have a header/footer added, will have a Bates number inserted for reference, and then be saved with a specified name and at a chosen directory.

This sample processes PDF files in the following way:

1. Extract range(s) of pages from the source PDF file and insert the extracted pages into a new PDF file. (To specify page ranges, use ';' to delimit different page ranges, use '-' to delimit starting page and ending pages for a page range. Use a single number if a page range contains a single page. For example, "1;3-5" indicates extracting page 1 and pages 3 to 5. In this notation, page numbers start at 1.)
2. Add a header to all the pages in the new PDF file.
3. Add a footer to all the pages in the new PDF file.
4. Stamp each page of the newly-created PDF file with a sequential Bates number, with the starting number as specified in the Bates Num field of the files_win.csv or files_mac.csv file.
5. Save the newly-created PDF file with the specified name and path.

All the values of the required parameters are stored in a CSV file, files_win.csv on Windows and file_mac.csv on Mac OS.

Usage

To run the sample on Windows, copy the files_win.csv, test.pdf and test1.pdf to C:\. On Mac OS, copy the files_mac.csv, test.pdf and test1.pdf to the root directory. (Make changes to the code and the content of files_mac.csv, if necessary.) When you select Civil Discovery from the menu item Advanced > Acrobat SDK, the plug-in processes the PDF files test.pdf and test1.pdf based on the information provided in the CSV file. The resulting files will be named test_pi_result.pdf and test1_pi_result.pdf for test.pdf and test1.pdf, respectively.

Note: Will not work in Adobe Reader.

DMSIntegration

Location

Plugins/Samples/DMSIntegration

Description

Demonstrate how a DMS system can integrate with Acrobat. This sample demonstrates the following three steps needed for DMS systems to integrate with Acrobat:

1. Write an ASFileSys for the DMS system.
2. Take over the File Menu - Open menu item and tool bar Open button.
3. Take over the File Menu - Save As menu item.

DdeServer

Location

PluginSupport/Samples/DdeServer

Description

Demonstrates how to establish communication between an external application and an Acrobat plug-in using DDE. This allows a plug-in to expose functionality to external applications that are not present in the standard IAC interfaces.

Usage

A sample client application is provided with the DdeServer plug-in. The application communicates with the plug-in to obtain the page number associated with the active page view and to add a text annotation to the page associated with the active page view. Acrobat must be running for the client application to function correctly.

Note: This plugin will not work in Adobe Reader since it accesses annotations. It is possible to use this technique to communicate with an Adobe Reader plugin.

Note: Windows only.

DocSign

Location

Plugins/Samples/DocSign

Description

Demonstrates how to use the PubSec API. It is more a "skeleton" plug-in than a fully functional sample, since it has no encryption library. The plug-in "cheats" at the points where a real one must not: when it comes to generating public/private key pairs, storing them, using them to encrypt data, etc. But it is presumed that a plug-in author knows this and only needs help with hooking in to Acrobat. That is what this skeleton sample plug-in does.

This sample was updated for the Acrobat 8 SDK release to demonstrate how to add a custom signature appearance.

Usage

To use DocSign, change the default signature handler in Acrobat Preferences to the DocSign handler. After that, DocSign will be used through the same user interface points as the standard signature handler.

Implementation details

This sample does not provide true digital signatures. It provides a skeleton of a PubSec API-based plug-in to which developers must add their own encryption routines.

Embed3DData

Location

PluginSupport/Samples/Embed3DData

Description

Demonstrates how to programmatically create a 3D annotation in a PDF file. For more information about the process, see the chapter titled "Creating 3D Annotations" in the Developing Plug-ins and Applications Guide (`plugin_apps_developer_guide.pdf`)

Program inputs:

- U3D file
- 3D JavaScript code file (optional)
- Poster image PDF file (optional)

The program output is a new PDF file with the 3D annotation.

Implementation details

1. Create a new PDF file with one page.
2. Add a 3D annotation to the page.
3. Specify dictionaries with key-value pairs in the 3D annotation.
4. Embed the 3D data.
5. Embed the optional JavaScript code.
6. Specify key-value pairs in the annotation dictionary.
7. Specify the optional Activation key-value pair.
8. Specify the 3DV (default initial view) dictionary.
9. Create an annotation appearance from the poster file if required. The code shows two possible ways to create a form XObject for the 3D annotation's appearance. The annotation appearance can be used for an external poster file as well as for an internal text poster.
10. Save the PDF file. The output PDF file works with Acrobat and Adobe Reader 7.0 or later.

Usage

Modifiable settings in the code:

- Path to the U3D file. This is specified by setting `gsDataFilePath`.
- Path to the JavaScript file. Use `gsJsFilePath` to specify this.

- Poster option, which is either an external image PDF file or an internal PDF form XObject. Specify the path to the poster file (`gsPosterFilePath`). If you are using an internal poster you also need to set `gbUseExternalPosterFile` to `FALSE`.
- Activation option. When opening the page, show the poster or the default initial view from the U3D data. Set `gbShowDefaultViewWhenOpenPage` to `TRUE` if you want to show the initial view of the 3D model.
- Names of the 3D view. An external name (`externalViewName`) can be used as a user interface. An internal name (`internalViewName`) is used to refer to the view from other objects.

Note: This plug-in will not work in Adobe Reader.

ExternalView

Location

PluginSupport/Samples/ExternalWindow/ExternalView

Description

Simple application that exercises the IAC interface exposed by the ExternalWindow plug-in.

The application communicates with the plug-in through Windows Messaging to have the plug-in open a PDF document in the main window of our application.

Note: Windows only.

ExternalWindow

Location

PluginSupport/Samples/ExternalWindow

Description

Demonstrates an approach for displaying an active view of a PDF document in the window of an external application. Select Open PDF in External Window from the Acrobat SDK submenu to run the plug-in.

Usage

A sample client application is provided with the Windows sample that passes a handle to its main window to the plug-in. The plug-in proceeds to display a PDF file in that window.

Note: Windows only.

ExtractBatesInfo

Location

Plugins/Samples/ExtractBatesInfo

Description

Demonstrates how to retrieve Bates number related information added by Acrobat 8 and above.

Usage

With a document open, select the "Extract Bates Info" menu item from the "Acrobat SDK" submenu.

PDFBinder

Location

PluginSupport/Samples/PDFBinder

Description

Shows how to programmatically implement To-PDF file conversion functionality. It converts multiple files into PDF using AVConversionToPDFHandler and binds the PDFs into one PDF file using PDDocInsertPages. It is designed to be executed from a menu; however, it can be run without a graphical user interface, so it can be executed conveniently from other programs using C IAC, Visual Basic IAC, or JavaScript to meet your enterprise workflow needs.

Most of the test files in the project's Testfiles folder are originally from *Acrobat 5: Classroom in a Book* with the copyright by Adobe Systems Incorporated and all rights are reserved. The files include: Afables1-3.pdf, Afables4.tif, Afables5.jpg, Afables5.jpj, Afables10-11.xml, Afables12-13.htm, Afables14.pdf, Afables15.txt, and Introduc.jpg. The other three files, bitmap1.bmp, bitmap1.jpg, and giffile.gif, were made by the Adobe Acrobat SDK team.

Usage

This sample adds a PDF Binder under the Acrobat SDK submenu. There are two ways to execute it:

- Click the menu item to run the program. This method uses a fixed file location that is hard coded (C:\PDFBinder\ on Windows, /PDFBinder/ on Mac OS).
- Press the Shift key and click the menu item. This method allows you to choose a folder where files to be converted are located.

When the hardcoded file location is used, you must put the files to be converted in the location before running the plug-in. A set of test files are provided in the project's Testfiles folder. You may store the files in another directory/folder, and specify the location in the code in the string variable PDFBinderFolder. An output file PDFBinderOutput.pdf in the same location is created when the program succeeds. A text log file PDFBinderLog.txt in the same location records the process and results. You can set bEch = false in the code to turn off any display on the screen. This is necessary when you call the menu function from within other programs written in C IAC, Visual Basic IAC, or JavaScript.

Implementation details

An optional file filter is used to pre-process the files. The filter allows only the files with predefined types to be processed. You can change the file type list if you like. Using a filter can ensure a smooth automation process, since you can put the file types to be tested in the filter. To turn off the filter, you can set gPDFBinderFileFilter = "" or do not define USE_FILE_FILTER.

Note: Will not work in Adobe Reader.

ProgressMonitor

Location

PluginSupport/Samples/ProgressMonitor

Note: Mac OS only.

Description

ProgressMonitor demonstrates how to create and use a custom progress monitor.

Usage

The plug-in can be exercised through the Acrobat SDK submenu's Progress Monitor Demo menu item.

Implementation details

The sample first acquires its progress monitor (a callback structure) through the method
`ASProgressMonitor GetProgressMonitor (void **clientData);`

This is a common approach for acquiring a progress monitor. The client will be returned the callback structure and the data (void*) that MUST be passed to each callback.

To demonstrate passing a progress monitor to one of the API methods, the sample calls `PDDocCreateThumbs` for the active document. Because the API method was written before the `setText` callback was added to the progress monitor, the code must make that call manually. `PDDocCreateThumbs` must also be passed a custom `ASCancelProc` rather than the default returned by `AVAppGetCancelProc`. This is required to allow the user to cancel the operation through the Cancel button in the dialog box.

The sample also demonstrates how to interact with any progress monitor when undertaking an extensive processing task that would otherwise freeze the user interface of the application. To demonstrate this, it counts the number of Link annotations on each page of the active document.

Note: While this issue is largely ignored by the sample, clients cannot expect a progress monitor to have implemented all the callbacks in the structure. Clients should verify that a callback has been implemented before attempting to call it.

Note: Will not work in Adobe Reader due to a call to `PDDocCreateThumbs`. The basic ProgressMonitor mechanism will work in Adobe Reader.

RplcFileSystem

Location

Plugins/Samples/RplcFileSystem

Description

Demonstrates how to create a custom file system. The sample is basically a wrapper around the default file system. The point is to demonstrate the mechanism for adding a new file system without becoming bogged down with complex implementation considerations.

Usage

The replacement file system can be exercised by selecting the Replacement File System menu item on the Acrobat SDK submenu.

Implementation details

The sample only implements the callbacks supported by the default file system.

SecurityHandler

Location

PluginSupport/Samples/SecurityHandler

Description

Demonstrates how to create a simple custom security handler. The new security handler takes the password, adds 1 to the ASCII value of each character and stores that in the file.

Usage

When you choose the File > Properties... menu item, in Document Properties dialog, select Security tab, "SDK Security Handler" appears as one of the options in the Security Options dropdown list. Selecting Change Settings allows you to restrict the operations that can be performed on specific objects in the PDF document.

Implementation details

The sample supports the extended encryption model first implemented in Acrobat 5.0. This model provides a finer level of control over the contents of PDF documents.

SelectionServer

Location

PluginSupport/Samples/SelectionServer

Description

Demonstrates how to implement a minimal selection server. The SelectionServer plug-in implements a selection server to handle images. The image selection tool allows users to select images by calling `AVDocSetSelection()` with a selection type of "Image". The selection server functionality is limited to getting, showing, and losing a selection.

Usage

The plug-in registers an Image Selection tool that allows users to select image XObjects on the page.

Installs the Advanced > Acrobat SDK > Image Selection Tool menuitem and the associated AVToolButton. You can use either to toggle the state of the selection server.

Note: Will display various messages using the SnippetRunner plug-in HFT if the SnippetRunner is installed and the CommonInterfaceFX.swf running.

Note: Will not work in Adobe Reader.

ShowPermissionsSDK7

Location

PluginSupport/Samples/ShowPermissionsSDK7

Description

Calls the PDDocPermRequest function to check operation permissions for the active PDF file under the current Acrobat product. The results will be saved to text files. This sample is a plug-in developed with Acrobat 7 SDK, and it can be Reader-enabled.

This plug-in will add a Show Permissions item under the Tools menu. When clicked, it will save the operation permission list to a plain text file and a tab-delimited text file in the location the user selects. The output's file names are the PDF file name plus -permissions.txt or .xls. The tab-delimited file can be opened by Microsoft Excel.

Acrobat products have different versions (7.x, 8.x) and variations (Adobe Reader, Acrobat Standard, Acrobat Professional, Approval, and so on), and a PDF may have a certain level of Reader-enabled usage-rights, so this plug-in can provide useful information about the Acrobat functionality, API availabilities and the PDF-permitted operations.

Stamper

Location

PluginSupport/Samples/Stamper

Description

Demonstrates how to implement an AVTool and an annotation handler. It also demonstrates use of the AVUndo API. It is accessed from the Acrobat SDK submenus' Stamper Annotations menu item.

Usage

Stamper annotations can be added to documents by selecting the Stamper tool and dragging a rectangle that defines the desired boundary for the annotation.

Note: Will not work in Adobe Reader.

Starter

Location

PluginSupport/Samples/Starter

Description

A plug-in template that provides a minimal implementation for a plug-in. Developers may use this plug-in as a basis for their own plug-ins.

Transparency

Location

PluginSupport/Samples/Transparency

Description

Demonstrates how to detect and modify the opacity of transparent elements in a PDF document.

Usage

The plug-in installs the Transparency menu item on the Acrobat SDK submenu. If the active document contains a transparent image, form, or path elements, activating the menu item displays a dialog box where users can manipulate the opacity of those elements. Two sample documents are provided in the `Transparency test files` folder to test the functionality of the plug-in.

Note: Will not work in Adobe Reader.

UncompressPDF

Location

Plugins/Samples/UncompressPDF

Description

A utility that removes all compression from the page content and form XObject streams within a PDF document. The sample illustrates how to implement an `AVConversionFromPDFHandler`, an object that is used to allow users to save PDF documents in a format other than those supported by Acrobat.

Usage

The functionality of the sample is accessible through a file type filter in the File > Save As dialog box. The filter name is "Uncompressed PDF Files". When any document is saved as that file type, the compression is removed from all of the document's page content and form XObject streams.

Implementation details

The sample uses the PDF Consultant framework to locate all of the page content and form XObject streams in the document.

Note: Will not work in Adobe Reader.

WeblinkDemo

Location

PluginSupport/Samples/WeblinkDemo

Description

Demonstrates how to register a new Weblink driver with the Weblink plug-in. This could be used to communicate with a web browser or application that is not included in the standard list, or to send information to the Netscape browser through a different interface than the standard Weblink driver.

Usage

The sample simply highlights the interaction between Acrobat and a Weblink driver. To this end, each callback writes a message to the Common Interface window indicating that it has been called.

WordFinder

Location

PluginSupport/Samples/WordFinder

Description

Demonstrates how to use the PDWordFinder methods to extract text from a PDF document.

Usage

The functionality of the plug-in is accessed through the menu items on the Acrobat SDK > Word Finder submenu. Create Page Map builds a page map (the root file name with an extension of ".map") that contains the offset from the start of the file for each word in the file. Once this is created, the user can use the Find Word By Word Offset menu item to display and highlight a word in the file, given its offset. On platforms with ADM APIs, the user is presented with an ADM dialog box in which to input a word offset; on UNIX (no ADM APIs), the offset is hardcoded into the plug-in source and no dialog box is presented. The default value is set as 1 and the first offset is always highlighted. The source can be altered to demonstrate different offset values being highlighted.

Implementation details

This plug-in can also be compiled to build page maps based on character offsets instead of word offsets by defining the CHAROFFSET symbol.

ADMCollection

Location

JavaScript/Samples/UI

Description

This primary JSADM sample contains five JSADM dialog boxes. It shows how to create and use various JSADM controls, including the button, list box, hierarchy list, text field, radio button, check box, and popup.

Usage

Open this PDF sample file and click the buttons to pop up the JSADM dialog boxes. Fill in text values or select items as instructed, and check the feedback in a pop up window. You can also click a button to pop up the JavaScript console window to check the feedback, but the console is not available in Adobe Reader for UNIX. To learn about the implementation, you can look at the JavaScript code inside the PDF file through the Acrobat Advanced menu.

AddSignature

Location

JavaScript/Samples/OutsidePDF

Description

Shows how to programmatically sign a PDF document using a predefined digital ID file. The JavaScript code includes all the digital signature information used to sign the document, except the path and password for the digital ID file.

To use this sample file, copy the `sdkAddSignature.js` file to the `JavaScript` folder under Acrobat, or to the user's `JavaScript` folder. When you restart Acrobat you should see a new item `Add My Signature` under the `Advanced` menu.

When you are ready to sign a PDF document, click the newly added `Add My Signature` menu item. After you input the platform-independent path and the password through a dialog box, the program creates a digital signature field in the top left corner. The path and password are valid in an Acrobat session, so you can continue to sign more documents in the session without the input dialog box. If you change the JavaScript code to specify the path and password for the digital ID file to be used, then when you click the menu item, the program will automatically sign PDF documents without requiring the UI. A digital signature file (`DrTest.pfx`) is provided with the sample. To use it, put it in a folder, and specify the proper `Dlpath` (for example `/C/DrTest.pfx`). The password is "testpassword".

It is also possible to execute the JavaScript code to sign a PDF document from another JavaScript program, or from a plug-in, Visual Basic (using interapplication communication), or Visual C++ program through the function `ExecuteThisScript`. See the source code for more information.

Beginning with Acrobat 7.x, in order to execute the security-restricted method through a menu event in this sample, the item labeled `Enable Menu Items JavaScript Execution Privileges` under `Edit > Preferences > General > JavaScript` must be checked. An alternative way is to wrap up the security methods used in the code through a trusted function. For details and examples, see `app.trustedFunction` in the *JavaScript for Acrobat API Reference*.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

AddToolButton

Location

JavaScript/Samples/UI

Description

This folder level JavaScript sample, AddToolButton.js, demonstrates how to add a toolbar button with the user-specified icon image file and JavaScript code.

Usage

The sample will add a menu item of Add Tool Button... under the Tools menu. Click it to display a JSADM dialog box. You must input a device-independent path to specify a PDF or image file (e.g. JPEG file) as an icon image file. The image should have a size of 20 X 20 pixels. Two sample image files, SampleIcon1.jpg and SampleIcon2.jpg, are included in the sample folder.

AnnotSample

Location

JavaScript/Samples/OutsidePDF

Description

Folder level JavaScript code to exercise the annotation APIs useful in reviewing workflow. It can be used with a rights-enabled PDF document in Adobe Reader as well as regular PDF documents in Acrobat.

Usage

This folder JavaScript file adds a new menu item under the Tools menu. It will trigger a JSADM dialog box to show the following functions:

- Set annotations as read only or editable
- Import annotations from a local FDF file
- Export all annotations to a local FDF file
- Export editable annotations to a local FDF file

You can try the functions while creating or modifying annotations in the PDF document.

To run the sample, you need to specify a path in the dialog box for a data repository in your environment. The path must be a safe path and in a platform-independent format, such as
`/c/test/myAnnotDataFile.fdf`.

A test file, ReaderEnabledSample.pdf, with Reader-enabled usage rights is provided in the sample folder.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

AnnotatedWords

Location

JavaScript/Samples/OutsidePDF

Description

This script returns to the console the words covered by highlight annotations. The script can be adapted to output the text elsewhere, such as to a file. It can also be edited to include other text-related annotations, such as underline or cross-out.

This script is limited to single-column, left-to-right, top-to-bottom, horizontal, non-overlapping text. Vertical text, text bound to a shape, right-to-left text, etc., may not work.

Annotated text is reported annotation by annotation in the order that the annotations were applied to the doc (appear in the annotations array), not in reading order or any other word order on the page.

This script uses `doc.getPageNthWordQuads()` to obtain its results. As such, it cannot report partial words under an annotation. Instead, any word that is partially covered by an annotation is included in the output. Additionally, line spacing, document origin, font size and other factors may affect how closely an annotation's quads and a word's quads overlap. For some documents, the results may not be accurate. Please see in-line comments for more information.

Usage

This JavaScript adds a menu item to the Comments menu: ACROSDK: Copy Highlighted Text to JS Console. The script will work on the active document and output words covered by highlight annotations to the console in a page-by-page list.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

ConvertTime

Location

JavaScript/Samples/InsidePDF

Description

ConvTime.pdf is a JavaScript sample that demonstrates how to convert the PDF date format to a JavaScript date object and back again. It also shows how to display the JavaScript date in various formats using utility methods.

Usage

There are three operation groups with instructions. Click the buttons and check results shown in text fields. You can convert the PDF date to a JavaScript date object with various formats, input your own date in the PDF format to check the conversion result, and convert the JavaScript date back to the PDF format.

DeleteNoCommentPages

Location

JavaScript/Samples/OutsidePDF

Description

DeleteNoCommentPages is a folder-level JavaScript code that can be useful in review workflows. It is similar to the Acrobat 6 Summarize Comments option which deleted pages without comments as it summarized. In Acrobat 7, this option was removed. This JavaScript allows you to simulate the Acrobat 6 option in other viewers.

Usage

This JavaScript adds a menu item to the Document menu under the DeletePages item. To run the sample, select Documents > Delete Pages Without Comments.

This script provides a status message (alert) upon completion. It can be removed for silent operation, such as batch processing. The processed document is not saved.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

Eliza

Location

JavaScript/Samples/SOAP

Description

The Eliza.pdf sample provides an Acrobat forms front end to a psychoanalysis web service. The sample uses a single API call to the service to get a string of text. The service is an implementation of the Eliza computer psychologist.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

EmbeddingFormData

Location

JavaScript/Samples/InsidePDF

Description

Demonstrates that a searchable database can be embedded inside a PDF form document, using the Doc's data object methods and other objects and methods. The code in this sample is located inside a PDF document, but it can be modified to be folder-level code to work with other PDF form files.

Usage

Some form data are already embedded in the PDF document, and you can click a form data entry under the Form Data List bookmark to retrieve the data. You can delete a bookmark to remove the data entry. There are buttons for you to reset form fields, to add or modify a set of data, and to search for certain form data. Click the Help button to get to the second page for detailed instructions.

Note: Will not run in Adobe Reader.

EventState

Location

JavaScript/Samples/Multimedia

Description

Demonstrates two ways for event listeners to have a local persistent state. It is particularly helpful to developers in writing multimedia event listeners. The boxes in the top row are ScreenAnnots with event listeners that log events to the list boxes below. Click each one to start logging, then move the mouse around among them and click some more to watch the events being logged.

The main functions are document-level JavaScript code. The two ScreenAnnots on the left use local variables and nested scope, and the two on the right use properties in the event listener object.

System Requirements. Acrobat 6.0 or later. The machine should be ready for multimedia play - sound card, speaker, proper system settings, and a multimedia player, such as Windows Media Player, QuickTime, or Flash.

GetStockPrice

Location

JavaScript/Samples/SOAP

Description

Provides an Acrobat forms front-end to a service that provides the stock price of a specified company (specified by stock symbol). It uses the SOAP.request method to connect to the web service and call the server's method.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

GoToBookmark

Location

JavaScript/Samples/OutsidePDF

Description

Provides a utility for users to get to a bookmark in a PDF document in Acrobat. If found, it executes the bookmark action defined in the PDF file. Usually this results in a page view, but other bookmark actions are possible.

The function `GoToBookmark`, demonstrated in this sample file, could be used, for instance, in accessing Help information in PDF documents. A help function could call this JavaScript method, specifying the PDF Help document and the bookmark to be found. The PDF page referenced by that bookmark would then be shown if the search succeeds.

To use this sample file, copy `sdkGoToBookmark.js` file to the JavaScript folder under Acrobat, or to the user's JavaScript folder. When you restart Acrobat you should see a new item `Go To Bookmark` located under the Tool menu in both Acrobat viewers and in Adobe Reader. Click it to get a dialog box to fill in your search criteria.

The input string is the full name of a bookmark to be found. The search is case insensitive. For example, if you open this Samples Guide PDF file, select the `Go to Bookmark...` menu item, enter the string `"GoToBookmark"`, and click OK, you will go to the beginning of this section because that heading is the first bookmark with the given name.

You may also specify the hierarchy level of your search in various ways, as in these examples:

`SDKJSSnippets` - Gets the first match in any level

`"Guide to SDK Samples:JavaScript Samples:Inside PDF:SDKJSSnippets"` - A completely specified hierarchy in which each token is one level down from the previous.

`"Guide to SDK Samples:*:SDKJSSnippets"` - A wildcard hierarchy in which "*" means there may be any number of levels there, including no level between.

The search process may be time consuming for a PDF document with a large number of bookmarks. To cancel the process, press `Esc` on Windows or `Command-period` on Mac OS. A progress bar is implemented using the thermometer JavaScript object.

JSCollection

Location

JavaScript/Samples/JSCollection

Description

A collection of JavaScript snippets organized by function and fully indexed for easy access. It includes sections for Acrobat forms(field manipulation, data validation, etc.) and for documents (bookmarks, navigation, etc.). This collection provides a range of basic JavaScript samples which can be cut-and-pasted into a PDF document to perform basic tasks or to help build larger workflow solutions.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

JavaScriptSnippet1

Location

JavaScript/Samples/InsidePDF

Description

Includes several JavaScript snippets that demonstrate objects and methods introduced in Acrobat 7.0. Each page in this sample document contains an independent piece of JavaScript sample code. The availability of JavaScript objects depends on the viewer type, platform, and code location, so some snippets have certain restrictions. For example, Text-To-Speech is a Windows-only sample; and Use of Template, Add Links, and Metadata do not work on Adobe Reader.

Contents:

- Execute JavaScript - assigns text input of JavaScript code to a button action and executes the input code with the button.
- Popup Menu - creates a popup menu.
- Metadata - shows a quick way to get document metadata in XML-formatted text.
- Full screen - displays a PDF document as an automatically-progressing, full-screen slideshow.
- Thermometer - displays a progress bar at the bottom of the Acrobat viewer window.
- Add Links - creates new links at specified words in a page.
- Text-To-Speech - presents a simple sample to speak the user's input text.
- OCG - demonstrates controls for hiding or showing optional content groups.
- Calculator - presents a functional calculator made by Acrobat forms with JavaScript.
- Show Paragraph - shows text blocks one-by-one at the touch of a hidden button, as in a PDF presentation.
- Hide Fields - shows or hides fields as might be used in a PDF presentation.
- Printing - controls document printing through JavaScript print parameters.
- Use of Template - adds a new page using a predefined page template.

Due to the functionality restriction, the following snippets won't work on Adobe Reader:

- Metadata
- Add Links
- Text-To-Speech
- Use of Template

There are annotations on each page in the file. Typically, the notes with the "Help" icon (question mark) provide help to users in running the snippet and the notes with the "Comment" icon (word balloon) are hints about how the snippet is implemented. Most of the JavaScript code is attached; users can look at it and modify it to experiment.

Note: Will not run in Adobe Reader.

OCGApiSample

Location

JavaScript/Samples/InsidePDF

Description

Demonstrates JavaScript APIs for PDF optional content groups. The code is embedded as document level JavaScript and executed through a JSADM dialog box.

Usage

To run the sample, open the JavaScript console as well as the Layer panel, then click the button in the PDF document to open a dialog box. There are eight buttons for you to try. Check the results on the Layer panel, document window, and JavaScript console.

Note: Will not run in Adobe Reader.

PresentationMonitor

Location

JavaScript/Samples/OutsidePDF

Description

Creates a set of tools to monitor the progress of a presentation using PDF slides.

Copy `sdkPresentationMonitor.js` to the `JavaScript` folder under `Acrobat`, or to the user's `JavaScript` folder. Then you should see a new item `Presentation Monitor...` under the `Advanced` menu. But before clicking the menu item, open a PDF file for the presentation (it is generally a PDF file including up to 30 slides). Now click the menu item to get a dialog box to enter the number of minutes you plan to use for the presentation. After that the monitor shown in the top of the slide page will start. When you go through the pages, check the following tools:

- A message showing number of pages untouched.
- A message showing time left.
- A time progress bar.
- A set of page icons:
 - The page icons with the different colors can indicate which page is current, and which pages have been visited.
 - When the mouse enters/exits a page icon, the page image will be shown/hidden in the top left corner.
 - Click a page icon to go to that page.
- A check box (the second one in the top-right corner): check/uncheck to show or hide the time bar and page icons.
- A quit button with "X" (in the top-right corner): click to quit the monitor tool.

Note: While the Thermometer is available in Adobe Reader, this sample will not run there due to use of methods not available.

PresentationNote

Location

JavaScript/Samples/OutsidePDF

Description

Creates a temporary note on top of the front PDF file to show the amount of time before a presentation begins.

Copy `sdkPresentationNotes.js` to the JavaScript folder under Acrobat, or to the user's JavaScript folder. Then you should see a new item Presentation Note ... under the Advanced menu. But before clicking the menu item, open the PDF file containing the presentation. Then click the menu item to get a dialog box. Enter the number of minutes before the start of the presentation and click the OK button to close the dialog box. The amount of time until the presentation begins will display and the time will be constantly updated until the specified time period is over. The display will last 10 seconds more after the end time, then go away. You may click the menu item again at any time to stop and remove the display.

SOAPCollabSample

Location

JavaScript/Samples/SOAP

Description

A folder level JavaScript sample that demonstrates how to implement a SOAP-based commenting collaboration store for Acrobat. It provides the minimum client-side code to connect to a web SOAP HTTP service while taking advantage of the built-in Acrobat commenting collaboration functionality. The collaboration functionality in Acrobat includes a few functions in the Annot and Collab classes, a Reviewing panel in Acrobat Preferences, and several online review tool buttons.

See the comments in the file for more information. In addition, the document *Acrobat Online Collaboration: Setup and Administration*, contains further details regarding what is involved in setting up a SOAP-based collaboration server using `sdkSOAPCollabSample.js`.

A public web service owned by a third party company is used for testing this sample. The operation of the sample code depends on the web server's availability and performance. Go to <http://www.ensemble-systems.com/pdfsamples/index.html> for more information.

To use the sample, copy `sdkSOAPCollabSample.js` into your JavaScript folder. Open the Edit > Preferences > Reviewing panel, where you will see a new SOAP-based collaboration server type. Select it and input the Server Settings as: `http://m43.ensemsys.com/jboss-net/services/PDFCollaboration?wsdl`. Then open a PDF document in your Internet browser to perform an online review. When the PDF document is loaded, the annotations stored in the repository on the server will be uploaded, if any exist. You can add, modify, and remove annotations, and click the online review buttons to send or upload. When the PDF document is closed, the updated annotation data is automatically sent to the server.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

ScriptEvents

Location

JavaScript/Samples/Multimedia

Description

This Acrobat multimedia sample demonstrates JavaScript commands sent from Windows Media Player, QuickTime, and Flash movies. This is a sample showing how to write JavaScript event listener functions to send out movie commands from a movie object, and interpret movie commands as you wish. It is a Windows-only sample, though the movie using QuickTime also works on Mac OS.

Click on a movie in the left two boxes, and the movie will begin and write scripts to the bottom script window. Click on the empty Flash Player box and you'll see a dialog box in the window. Type in a string in the text field, and click the button below it; you'll see your string sent to the lower script window.

Main functions are set as document-level JavaScript. An AfterScript listener function was created to send out the movie command with a parameter which is the movie script.

System Requirements. Acrobat 6.0 or later. The machine should be ready for multimedia play - sound card, speaker, proper system settings, and a multimedia player, such as Windows Media Player, QuickTime, or Flash player should be installed.

Note: The Windows Media Player movie requires a codec that doesn't ship with Vista. You can find a version that supports this encoding here: http://www.voiceage.com/acelp_eval_eula.php.

SigningWorkflow

Location

JavaScript/Samples/InsidePDF

Description

Tutorial demonstrating how to sign a PDF by calling a key from the Windows keystore using JavaScript.

Usage

To see the tutorial, open SigningWorkflow.htm from browser.

SigningWorkflow.js contains the JavaScript code used in the tutorial.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

SilentPrint

Location

JavaScript/Samples/OutsidePDF

Description

Folder level JavaScript code to demonstrate the print APIs. It can be used in Adobe Reader as well as Acrobat. There are many print options that users may set up in the JavaScript print parameters. See the *JavaScript for Acrobat API Reference* and *Developing Acrobat Applications Using JavaScript* for further information about silent print functionality.

Usage

The sample will add Silent Print under the File menu. Click it to print the current document to the default printer without displaying the Print dialog box.

TextExtract

Location

JavaScript/Samples/OutsidePDF

Description

A folder level JavaScript sample that demonstrates how to extract the text in PDF page contents and save it to a local file. The JavaScript `Doc.getPageNthWord` method is used to get the words one by one from the current page, and then a data object is created and exported.

Usage

The JavaScript code will add an Extract Text ... item under the Document menu. When the new menu item is clicked, you can select a file to where you can save the text extracted from the current page. You can use Microsoft Word to view the text file.

Note: Requires Reader-enabled PDF document to run in Adobe Reader.

Toolbarbutton

Location

JavaScript/Samples/UI

Description

A PDF document demonstrating how to add and remove tool bar buttons using JavaScript APIs. The `addToolButton` method takes an Icon Stream Generic Object. In this sample, the stream data is hard-coded as a hexadecimal string to represent the alpha, red, green and blue channels. See the code associated with the Add button icon for details.

In this sample the `addToolButton` method is called from the PDF document, so it will be attached to the document, and removed when the PDF document is closed. There is another SDK sample to show how to add a button to Acrobat tool bars.

Usage

Click the buttons in the PDF document to add or remove the tool button.

Trophy

Location

JavaScript/Samples/Multimedia

Description

This Acrobat multimedia sample demonstrates the use of JavaScript to cue up two media players and then start them playing simultaneously.

Click the Play button to see the two movies playing. The movies have no sound. When you click the Play button, JavaScript code will open each player and install an afterReady event listener in each one. When all players have reported afterReady, the code calls the play() method on each player. This way, the players start within a fraction of a second of each other.

System Requirements. Acrobat 6.0 or later. The machine should be ready for multimedia play - sound card, speaker, proper system settings, and a multimedia player, such as Windows Media Player, Apple QuickTime, or Adobe Flash.

TwoPartInvention

Location

JavaScript/Samples/Multimedia

Description

This is a PDF document with sheet music you can click to play back. The music is a QuickTime file which contains a MIDI track. A C++ program using the QuickTime API was used to add a marker at each measure in the QuickTime file, and a script event at each measure and beat within a measure. When the music is playing, the script events trigger JavaScript code in the PDF document that outlines the current measure and beat and turns the page when needed. You can also click a measure for the music to jump to.

You can click About to learn the implementation details.

System Requirements. Acrobat 6.0 or later. The machine should be ready for multimedia play - sound card, speaker, proper system settings, and a multimedia player, such as Windows Media Player, QuickTime, or Flash.

Note: Will not run in Adobe Reader.

AddAnnotations

Location

IAC/mac/AppleScripts

Description

Demonstrates how to create and manipulate the annotation objects that are supported in the Acrobat AppleScript dictionary - text and link annotations.

To function properly, the script requires a PDF file with at least three pages. It places a text annotation on page 1, and a link annotation on page 2. The link annotation's destination is set to the last page of the document and then performed.

Implementation details

The text annotation type is actually comprised of two annotations. One is used for the annotation itself and the other is for the associated popup. When a text annotation is created using the Make Text Annotation command, its popup will immediately follow it in the annotations array of the page.

To successfully create link annotations, you must specify at least one of the link-specific properties, such as the destination page number or fit type, when the annotation is being created. For example:

```
set props to {destination page number: 0}
set linkAnnot to make Link Annotation with properties props
```

DistillerControl

Location

IAC/mac/AppleScripts

Description

Demonstrates two methods for converting PostScript files into PDF using the Distiller AppleScript interface. The first uses the default settings and the second sets a specific JobOptions setting.

ObjectProperties

Location

IAC/mac/AppleScripts

Description

Demonstrates how to work with various objects exposed through the Acrobat AppleScript interface including the application, document windows, documents, and pages.

PrintPage

Location

IAC/mac/AppleScripts

Description

Prompts the user to browse for a PDF file then prints the first page to the default printer.

RotatePages

Location

IAC/mac/AppleScripts

Description

Demonstrates low-level page manipulation. It also demonstrates processing all PDF files within a chosen folder.

The user is prompted to select a degree of rotation before browsing for a folder. For each PDF document in that folder, every page in the document is rotated by the specified amount.

SelectText

Location

IAC/mac/AppleScripts

Description

Demonstrates how to create text selections within a document open in the viewer.

WatermarkJsoAS

Location

IAC/mac/AppleScripts

Description

Demonstrates how to access the JavaScript object to execute `addWatermarkFromText` and `addWatermarkFromFile` methods. This sample mimics the Windows IAC sample `WatermarkJsoVB`.

AcroPDFInHTML

Location

IAC/win/HTMLSamples/AcroPDFInHTML

Description

A simple example of using the PDF control embedded in an HTML page. The <OBJECT> tag with a unique classid for AcroPDF ActiveX control is used to embed it. That is, classid="clsidCA8A9780-280D-11CF-A24D-444553540000".

This sample also demonstrates how to use the automation API on the control from client-side JavaScript.

Usage

Users must have Internet Explorer installed on their machine. To make the sample work properly, set the option to allow blocked content. For example, the user can click the security bar.

The sample includes four HTML files. Click the frameset1.htm file to start. A web page with frames will display in the browser. Input a URL to a PDF file and click the Go button to pop up the PDF ActiveX control. There are two buttons in the left frame, the user can click to move to the previous or next page. The sample code only works with Internet Explorer on Windows, but the approach can be adapted for other browsers.

AcrobatActiveXVB

Location

IAC/win/VBSamples/AcrobatActiveXVB

Description

Demonstrates how to use the Acrobat ActiveX control to embed a PDF document window with toolbars in a VB.NET application. It also shows how to use the ActiveX APIs to create user graphical interfaces to externally control the PDF window.

Acrobat provides a COM-based automation interface (AcroPDF.dll). You can add it to your project's references to add the control to your toolbox. Once you drag the ActiveX control to a window in your application, you can open a PDF document window with tool bars. You can also use the APIs provided by the ActiveX control to programmatically manage the window for opening a PDF file from a URL or a local disk, page navigation, setting display mode or style, hiding tool bars, and so on.

Usage

You must have Acrobat or Adobe Reader installed to run this VB.NET application. You can access a PDF document in a URL or open a PDF document on a local disk, and then use the tool bar in the ActiveX control or the buttons and dialog boxes implemented with APIs to control the PDF document window.

ActiveViewVB

Location

IAC/win/VBSamples/ActiveViewVB

Description

Demonstrates how to use the Acrobat automation interface to display an interactive view of a PDF document within a VB.NET application window.

Usage

This sample is an MDI application that opens PDF documents as if the application were rendering them itself. The user can perform a number of operations to edit the PDF document and manipulate the view.

ActiveViewVC

Location

IAC/win/CSamples/ActiveViewVC

Description

Demonstrates how to use the Acrobat automation interface to display an interactive view of a PDF document within a VC.NET application's window.

Usage

ActiveView is an MDI application that opens PDF documents as if the application were rendering them itself. The user can perform a number of operations to edit the PDF document and manipulate the view.

AdobePDFSilent

Location

IAC/win/VBSamples/AdobePDFSilentVB

Description

This Visual Basic sample demonstrates how to silently print to the Adobe PDF printer. It processes files from a specified input directory (the default is the application's current directory) and places output PDF files in a specified output directory (the default is the current directory). If a valid directory is not specified, output is directed to My Documents. Output file names are created from the original file name with a .pdf extension.

The application uses Windows printing and can process file types for which there is a registered application, that is, Print appears on the file's context menu. Note that the application will try to process a file that has an associated application for opening the file even if it does not have a print process associated with it. That is, the file's context menu has an Open item but not a Print item. This may generate an unhandled exception. To manage such occurrences, there is an array of file types (file extensions) not to process. The array is initially set to ignore three file extensions: .dll, .exe and .pdf. Others may be added. Files for which an associated application cannot be found are skipped (not converted) without notice.

Because this application is dependent on Windows printing and specific registry entries, successful conversion of specific files, types and locations will vary from machine to machine and user to user. Careful testing should be done to ensure desired results can be achieved.

Usage

The application (AdobePDFSilent.exe) can be invoked from the command line. One or two arguments are required for automated use. If one argument is specified, it is used for both the Input and Output directory paths. If two arguments are included, the first is used to set the Input directory path and the second is used for the Output directory.

To prevent the created PDF documents from opening an Acrobat viewer, unselect the View Adobe PDF Results option in the Adobe PDF print driver printing preferences.

BasicIACVB

Location

IAC/win/VBSamples/BasicIACVB

Description

A simple Visual Basic sample for interapplication communication. It includes the code to launch the Acrobat viewer, open a PDF file (C:\sample.pdf), and get simple information (number of pages).

BasiclacCS

Location

Description

A simple sample that provides the minimum code to use the interapplication communication in a C# application. It includes code to launch Acrobat, open a PDF file (C:\sample.pdf), and get simple information (number of pages).

BasicIacJsoVB

Location

IAC/win/VBSamples/BasicIacJsoVB

Description

This simple Visual Basic sample demonstrates how to use the interapplication communication (IAC) JavaScript object in Visual Basic applications. It includes the minimum code to create IAC objects and use their properties and methods. The program opens a PDF file (`C:\TestForm.pdf`) and gets some information about the document (number of pages, number of words, and number of form fields).

BasiclacOCXCS

Location

IAC/win/VBSamples/BasiclacJsoVB

Description

Demonstrates how to use the PDF ActiveX control in a C# application.

Usage

This sample displays two PDF windows within the sample form. Type in a URL or browse a local PDF file by clicking a Browse button. The file specified in the Address field will be displayed in the corresponding PDF window after you click Go. The two windows can display the same or different PDF files.

BasicIacVC

Location

IAC/win/CSamples/BasicIacVC

Description

Provides the minimum code to use interapplication communication (IAC) in a Visual C++ application. The sample first creates an Acrobat IAC PDDoc object, then tries to open a sample file `C:\sample.pdf`. If the file is opened successfully, it gets the number of pages and displays this number in the dialog box. Otherwise, it shows an error message.

DdeOpenVC

Location

IAC/win/CSamples/DdeopenVC

Description

DdeOpenVC demonstrates how to communicate with Acrobat through DDE interfaces.

Usage

The sample attempts to open a PDF document in Acrobat. The path "c:\test.pdf" is sent to Acrobat as the parameter of the FileOpenEx command.

Implementation details

DDE server executables must be running before external applications can initiate a conversation with them. The sample checks the following key in the registry to determine if Acrobat has been installed:

```
HCLM\Microsoft\Windows\CurrentVersion\App Paths\Acrobat.exe
```

If the key is found, the path stored under the key is launched.

DistillerCtrlVB

Location

IAC/win/VBSamples/DistillerCtrlVB

Description

Demonstrates how to use the Acrobat Distiller automation interface to convert PostScript files to PDF in a VB.NET application.

Usage

Click Select Input File to select the PostScript file that will be processed. The Visual Basic implementation also allows users to select a job options file to use when processing the file. The progress of the conversion operation is shown in the application window.

Implementation details

Both implementations receive progress updates from Distiller. This is achieved through the use of the `_PdfEvents IConnectionPoint` interface that the automation interface supports.

Visual Basic makes use of this interface rather easy - simply declare the Distiller application object with the `WithEvents` keyword, for example, `Private WithEvents pdfDist As PdfDistiller`

When the `WithEvents` keyword is used, Visual Basic provides the framework for the application to provide implementations for each event in the interface. The C++ implementation is more involved. The application must provide an implementation of the `_PdfEvents` interface and register it with Distiller using the `Advise` method on the `_PdfEvents IConnectionPoint` interface.

DistillerCtrlVC

Location

IAC/win/CSamples/DistillerCtrlVC

Description

Demonstrates how to use the Acrobat Distiller automation interface to convert PostScript files to PDF in a C++ application.

Usage

Click Select Input File to select the PostScript file that will be processed. The Visual Basic implementation also allows users to select a job options file to use when processing the file. The progress of the conversion operation is shown in the application window.

Implementation details

Both implementations receive progress updates from Distiller. This is achieved through the use of the `_PdfEvents IConnectionPoint` interface that the automation interface supports.

Visual Basic makes use of this interface rather easy - simply declare the Distiller application object with the `WithEvents` keyword, for example, `Private WithEvents pdfDist As PdfDistiller`

When the `WithEvents` keyword is used, Visual Basic provides the framework for the application to provide implementations for each event in the interface. The C++ implementation is more involved. The application must provide an implementation of the `_PdfEvents` interface and register it with Distiller using the `Advise` method on the `_PdfEvents IConnectionPoint` interface.

DistillerCtrlWMVC

Location

IAC/win/CSamples/DistillerCtrlWMVC

Description

Demonstrates how to use the Acrobat Distiller Windows messaging interface to convert PostScript files to PDF in a C++ application.

Usage

The sample allow users to select the PostScript file that will be processed. The user can also specify the level of interactivity that Distiller should use to determine the name of the output file.

Implementation details

An instance of the Distiller application must be running to receive the messages from the external application. The sample checks the following key in the registry to determine if Acrobat has been installed:

```
HCLM\Microsoft\Windows\CurrentVersion\App Paths\AcroDist.exe
```

If the key is found, the path stored under the key is launched.

External applications can initiate conversion operations within Distiller by sending it the WM_COPYDATA message. As the LPARAM parameter, you must pass a pointer to a COPYDATASTRUCT structure which contains (amongst other things) a DISTILLRECORD structure. The DISTILLRECORD structure contains the conversion parameters for the operation being performed.

For the WPARAM parameter, you can pass an HWND to which Distiller should report the status of each conversion operation. This response is also passed using a WM_COPYDATA message. When the specified HWND receives the reply message from Distiller, the LPARAM parameter contains a pointer COPYDATASTRUCT structure which contains a DISTILLRECORD structure. The param member of this structure will contain zero if the operation was completed successfully, or -1 if some error occurred.

ExecuteScriptIacVB

Location

IAC/win/VBSamples/ExecuteScriptIacVB

Description

Demonstrates how to execute JavaScript code by calling the AcroForm OLE ExecuteThisScript method in Visual Basic applications. The default code writes information about the active PDF document to a new PDF report file, then opens the report. The program provides a dialog box for users to modify, rewrite, and execute their own JavaScript code.

Usage

- Click PDF Document to open or change the active PDF file open in Acrobat.
- Click Execute to execute the JavaScript code. First try the default JavaScript code mentioned above, and check the PDF report in Acrobat. Close the report.
- Click Clear to clear the text window, then input your JavaScript to execute.
- Click Reset to restore the default JavaScript code.
- Click Help to check the help message.
- Click Close to quit the program. Acrobat viewer also quits if it is opened by this program or it has no PDF files opened.

The BasicIacVBJSO sample also shows how to execute JavaScript code from an IAC application.

FillFormCS

Location

Description

Demonstrates how to fill a form from a C# application using interapplication communication.

Usage

Copy the file `SampleForm.pdf` from the `FillFormCS` folder to `C:\`, then run the application. The form field `Name` will be filled with value `John Doe`.

FormsAutomationVB

Location

IAC/win/VBSamples/FormsAutomationVB

Description

Demonstrates how to create and manipulate form fields using the automation interface exposed by the Acrobat Forms plug-in in a VB.NET application.

Usage

Before running the sample you must copy the `FormsAutomation.pdf` file to the `C:\` directory. The sample creates a variety of form fields in the file. It is best to have Acrobat open and visible to see the form creation.

To run as a batch process, save the form and close the file in Acrobat. There is commented-out code in the sample that shows how to do this.

JSOFindWordVB

Location

IAC/win/VBSamples/JSOFindWordVB

Description

A JSObject sample that shows how to access JavaScript using Visual Basic. The approach is to get the JavaScript object from the PDDoc of a PDF file, then call JavaScript methods.

The application displays a dialog box in which the user can select a PDF file and input a word (in Roman characters only) to find. Clicking the Find Word button displays the page on which the word is first found and highlights the word. After each occurrence is found, the user is asked whether to continue the search. The final count of words found is displayed in the dialog box.

Implementation details

Since the JavaScript code runs slowly, it is not suitable for searching through a large PDF file.

JObjectAccessVB

Location

IAC/win/VBSamples/JObjectAccessVB

Description

Demonstrates how to access the JavaScript object model through the Acrobat automation interface.

Usage

Before running the sample, the following variables must be configured in JObjectAccess\JObjectAccessVB.vb:

- SOURCE_DOCUMENT: device-independent path to the source PDF document.
- DATA_FILE: device-independent path the data file.
- OUTPUT_FOLDER: device-independent path to the output folder.

RemoteControlAcrobatVC

Location

IAC/win/CSamples/RemoteControlAcrobatVC

Description

An MFC Windows program that controls Acrobat remotely through IAC (interapplication communication). It shows how to start IAC, create IAC objects, and call their methods. The program has an Acrobat menu that has the following items:

Launch

- App submenu: Show Acrobat, Hide Acrobat
- AVDoc submenu: Open, Close, Print PDF, Find Text in PDF
- AVPageView submenu: Display Page Number, Go to Next View, Go to Previous View, Go to Page
- Exit

The menu item Find Text in PDF simply finds and highlights the first occurrence of a word in Roman characters. Since a workspace with sample code has been set up, the user can easily add any other Acrobat IAC objects and methods to the program. The source file `RemoteControlAcrobat.cpp` contains information about the program, its limitations, and how to extend it in real programs.

SearchPdfVB

Location

IAC/win/VBSamples/SearchPdfVB

Description

Demonstrates how to communicate with the Search plug-in through its DDE interface. The DDE interface allows external applications to manage indices and initiate queries.

Usage

The SearchPDF application divides its functionality into two distinct operations; index manipulation and search queries. Users can add, remove, enable, or disable index files.

Users can also formulate a search query and specify a number of query options, including the query parser to be used and the maximum number of documents returned.

Implementation details

The application uses a C-based DLL to manage the DDE conversation with the Search plug-in. The interface provided by the DLL mimics the functionality exposed by the plug-in's DDE interface. The DLL (`ddeproxy.dll`) should be in the system directory or the directory where the Visual Basic program is located.

StaticViewVB

Location

IAC/win/VBSamples/StaticViewVB

Description

Demonstrates how to use Acrobat's automation interface to render the contents of a PDF page into a VB.NET application's window.

Usage

StaticView is an MDI application that opens PDF documents as if the application were rendering them itself. The user can manipulate the view of the PDF document through the toolbar and View menu.

StaticViewVC

Location

IAC/win/CSamples/StaticViewVC

Description

Demonstrates how to use the Acrobat automation interface to render the contents of a PDF page into a C++ application's window.

Usage

StaticView is an MDI application that opens PDF documents as if the application were rendering them itself. The user can manipulate the view of the PDF document through the toolbar and View menu.

WatermarkJsoVB

Location

IAC/win/VBSamples/WatermarkJsoVB

Description

A VB.NET sample that shows how to add watermarks to a PDF document programmatically. Two JavaScript methods, `addWatermarkFromFile` and `addWatermarkFromText`, are used through the JavaScript object in the IAC application.

Usage

Copy the test files, `SamplePDF01.pdf` and `ReceivedStamp.pdf`, to `C:\` and run the program. A stamp with a date and time will be added to the top left corner of the first page in the PDF document.

Due to security restriction enforced by Vista, exercise this sample in one of the two ways on Vista:

- (1) Run Acrobat with "administrator" privilege, then run the sample with "administrator" privilege.
- (2) Make sure Acrobat is not running. Quit Acrobat if it is. Run the sample with "administrator" privilege.

To run an application with administrator privilege, right-click on the application icon to bring up the context menu then select "run as administrator." Note that logging in with administrator capacity does not override the security restriction.

APILocator

Location

Tools/APILocator

Description

- Provides the ability to browse all APIs (C, OLE, JavaScript, and AppleScript) supported by Acrobat in a class hierarchical manner, and provides descriptions of those APIs.
- Links each API to its use in source provided by the SDK. By navigating to the API in the browser, a panel in the Sample Info tab will display a list of source files that contain that API, and the location of use. Clicking on a source file will display that file, clicking on a location will scroll to that location and highlight the API.
- Provides a search mechanism that searches across all four API flavors. Highlighting a search result also gives you the description and source API links, if any.

Usage

By clicking in the browser at the top of the window, you will find APIs in an object-oriented classification. Choosing one of the leaf nodes will display information in the bottom pane. The description of the API is from the related API reference. The source listing is from source included in the SDK.

You can also switch to the search panel. Here you can search across the APIs for a relevant interface. This allows you to either find the specific API you are looking for without knowing the specific name or object it is part of, or you can compare APIs across languages, to see what will best meet your needs.

Extra features in the search panel are: 1) clicking on a column header will sort by that column, and 2) by choosing any of the contextual menu options you can narrow or broaden the search as you see fit.

Note: Windows and Mac OS only.

Note: The Mac OS version requires Mac OS X 10.3.x or better. The Windows version is built using C#, and thus requires the Microsoft .Net Framework to be installed on your machine.

Note: Both platforms use data files for the information displayed. The Windows data files are in the 'data' directory located in the same directory as the application. The Mac OS files are internal to the application package. In both cases, if they are moved or deleted the application will lose functionality.

Note: On Windows, Copy is available in the context menu for the description and source panes.

Plug-in Wizard

Location

Tools/Visual Studio App Wizard

Description

Wizard to create plug-ins for Acrobat 8 with Visual Studio 2005

The Plug-in Wizard extends Microsoft Visual Studio 2005 to simplify the creation of Visual Studio C++ projects that create plug-ins for Acrobat 8.

Installation

If Visual Studio 2005 is installed when the Acrobat 8 Plug-in wizard installer is run, the installer will install the necessary files in the Visual Studio 2005 installation.

Usage

In Visual Studio 2005, create a new project by clicking on **New Project** or choosing the File > New > Project command. Select "Visual C++ Projects" in the Project Types pane. Choose Acro8PIWiz in the Templates pane. Enter a name and location for the project and click OK.

The first time you are using Acrobat 8 Plug-in wizard, you will be prompted to locate Acrobat 8 SDK header path. You can check "Do not show at startup" if you don't want to see the dialog next time you start 2005 Visual Studio. However, if you want to check whether you have specify the correct location for the headers at a later time, you can always click on the "Headers" button to bring up the dialog showing the header path.

The Plug-in Wizard will now start, offering you many high-level choices about the plug-in that will be created. Select those that you desire and the wizard will create the project and populate it with the necessary files.

Note: Replaceable functions can have many different return types, from void to built-in to structured types. Because of this, projects that use replaceable functions will draw compiler warnings as the variable used to hold the return value is not initialized. You should, of course, complete the replaceable function to do what you would like, and in the process initialize the return value to eliminate the compiler warning.

Note: Windows Only.

Reader-enabling Tools

Location

Tools/Reader-enabling Tools

Description

For detailed information on using the Reader-enabling tools, please see the PDF document entitled "plugin_apps_developer_guide.pdf" located in the Acrobat SDK at the following location:

Documentation/Introduction_To_SDK/plugin_apps_developer_guide.pdf

ShowPermissions

Location

Tools/ShowPermissions

Description

ShowPermissions uses the PDDocPermRequest function to check operation permissions for the front PDF, and saves the results to text files.

Since the Acrobat products have different versions and variations and the PDF may have certain level of Reader-Enabled usage-rights, this plug-in can provide useful information about the Acrobat functionality / API availabilities and the PDF permitted operations.

ShowPermissions.api will work with Acrobat 7 or later, and it has been Reader-enabled for use with Adobe Reader. Note that if you directly build the ShowPermissions plug-in sample in the SDK, then that plug-in won't be able to run in Adobe Reader.

Usage

Copy this file to the Acrobat or Adobe Reader plug_ins directory. The plugin will add a menu item titled "Show Permissions" under the Tools menu. When chosen, you can select a folder for the output, and then save the operation permission list to a plain text file and a tab delimited text file. The latter can be opened by Excel. The output's filenames are the PDF filename plus -permissions.txt or .xls.

ShowPermissions6.api is for use with Acrobat 6 or Adobe Reader 6.

Note: Windows only.

VerifyURLs

Location

Tools/VerifyURLs

Description

VerifyURLs is a tool that scans a document for all of the link annotations with "World Wide Web Link" actions and verifies that the URL associated with those annotations points to valid resources.

Usage

The functionality of the tool is accessible through the Advanced > Acrobat SDK > Verify web links in document menu item. If the user needs to specify a proxy server address and port the plugin uses Acrobat web file system, so it will use Acrobat's settings. The plug-in proceeds to scan all the link annotations within the active document. If one or more link annotations are found with "World Wide Web Link" actions, the plug-in attempts to verify that the links are valid.

Note: Mac OS X only.