

Acrobat 7.0.5 SDK




Guide to SDK Samples

November 8, 2005



Adobe Solutions Network — <http://partners.adobe.com>



Copyright 2005 Adobe Systems Incorporated. All rights reserved.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated. No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the Adobe Systems Incorporated.

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

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

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

Apple, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. PowerPC is a registered trademark of IBM Corporation in the United States. ActiveX, Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Verity is a registered trademark of Verity, Incorporated. UNIX is a registered trademark of The Open Group. Verity is a trademark of Verity, Inc. Lextek is a trademark of Lextek International. All other trademarks are the property of their respective owners.

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



Guide to SDK Samples

Introduction

The Adobe® Acrobat® products include extensive APIs for integration. The full installation of the Adobe Acrobat 7.0.5 Software Development Kit (SDK) contains a large number of sample applications, code snippets, plug-in projects, tools and scripts to demonstrate how to use these APIs.

This guide provides descriptions and implementation details for samples included in one or more installations of the Acrobat 7.x SDK, such as the Adobe Acrobat SDK for Windows®, the Adobe Acrobat SDK for the Mac OS®, the Adobe Reader SDK for Linux® and the web-based Exploded SDKs on the Adobe Partners web site. Note that not all samples in this guide are included in every installation of the SDK.

For instance, not all contents and samples provided in the full installation of each Acrobat 7.x SDK are included in the same version of the Exploded SDK. Specifically, Tools included with the full installation are not available in the Exploded SDK. Additionally, although the sample applications and plug-ins provided in the full installation include platform-specific IDE project files, only source files are provided in the Exploded SDK.

While much of the Acrobat APIs are platform independent, not all samples function on all supported platforms. If a sample does not function on a specific platform, it is excluded from the SDK installations for that platform. Additionally, Adobe Reader and Adobe Acrobat Standard expose subsets of the APIs of Adobe Acrobat Professional. Therefore, samples that do not function in Reader are not included in Reader-specific SDK installations and not all samples in the Acrobat SDK will function in Acrobat Standard. For specific information on developing with Reader, please see the "Adobe Reader SDK Development Overview" document included in the Adobe Reader SDK.

The sample descriptions in this guide are grouped into these categories: IAC (Interapplication Communication) Samples, JavaScript Samples, Plug-in Samples, and Other Samples (SaveAsXML). In some cases, platform and/or viewer limitations may be noted in the sample details.

NOTE: On Windows, the Acrobat 7 SDK uses an OS environment variable called **AcroSDKPIDir**. If **AcroSDKPIDir** exists, then Visual Studio uses the directory it points to as the destination for a compiled sample plug-in; if does not exist, no copy is performed. Setting **AcroSDKPIDir** is OS-specific and must be done before invoking Visual Studio. See *Acrobat SDK Plug-in Guide* for more information on this environment variable.

NOTE: On UNIX, the Adobe Dialog Manager (ADM) plug-in API methods are not exposed, rather ADM functionality is available only via Acrobat JavaScript. Plug-in samples provided in the Linux Reader SDK that were implemented with ADM API methods in their Windows and/or Mac OS versions were updated to exclude the use of ADM. For

some samples, the UNIX-specific non-ADM functionality is not described in this document, and the Release Notes or source files should be consulted for details.

New tools and samples in SDK 7.0.5

JavaScript Samples

- JSCollection
- SilentPrint

IAC Sample

- WatermarkJsoAS

New tools and samples in SDK 7.0

Tools

- The API Locator Tool
- ShowPermissions

JavaScript Samples

- ADMCollection
- AnnotSample
- EmbeddingFormData
- OCGApiSample
- ToolbarButton

IAC Samples

- AcrobatActiveXVB
- BasicIacCS
- BasicIacOCXCS
- FillFormCS
- WatermarkJsoVB

Plug-in Sample

- ShowPermissionsSDK7

Snippets

- ASBigFileSnip
- ASCabPutGetSnip
- ASChangeTempFileSysSnip
- AVPageViewToggleWireframeDrawing
- AVWindowMaximizeCurrentPageView
- ConvertFlate2JPXSnip
- CosDoc64Snip
- CosStream64Snip
- JPXColorSpaceExplorerSnip
- JPXPaletteExplorerSnip
- PDOCGChangeLockedStateSnip
- SmartPDPPageSnip
- UserPropertiesExplorerSnip

Improved/Updated for the Acrobat 7.0 SDK

- CreateContentXORSnip
- GetStockPrice
- PDCreateMasterOCGSnip
- PDOCConfigCreateSnip
- PDOCConfigExplorerSnip
- PDOCGExplorerSnip
- PDOCSetDefaultConfigSnip
- Visual Studio Plug-in Wizard

Tools

APILocator

NOTE: Windows and Mac OS only.

APILocator Functionality

1. 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.
2. 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.
3. Provides a search mechanism that searches across all 4 API flavors. Highlighting a search result also gives you the description and source API links, if any.

Use

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: The Mac version requires 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 files are internal to the application package. In both cases, if they are moved or deleted the application will lose functionality.

Plugin Loader

Plugin Loader allows you to not have to quit and restart Acrobat during plug-in debugging sessions.

NOTE: Not all samples are compatible with the Plugin Loader.

Install the PluginLoader plug-in in the Acrobat plug-ins folder, then launch Acrobat. A menu item is added to the **Advanced > AcrobatSDK** menu named **Load Plugin...**. When that menu item is selected, an open file dialog box appears. Choose the plug-in you want to load, and then click OK. The PluginLoader will attempt to load your plug-in, using the same initialization steps that Acrobat uses when loading plug-ins at startup.

Your interface should then be available for testing. When you want to fix your plug-in or load a different one, choose the same menu item (now called Unload Plugin), to unload your plug-in and clean up.

NOTE: Your plug-in won't get Acrobat initialization notifications, such as AVAppDidInitialize.

NOTE: Your plug-in must unload itself correctly in the PluginUnload routine or an unstable condition might exist. This means at least removing UI elements and unregistering for notifications of any sort.

NOTE: (Mac only) For PluginLoader to find the main routine for your plug-in, the routine must be exported. Standard Mac plug-ins don't do this. The easy way to do this is to change your plug-in's Export Symbols settings on the PPC PEF panel to "All Globals", and then relink the plug-in.

Reader-enabling Tools

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

Adobe Acrobat 7.x SDK/Documentation/Plugins/ReaderEnabling.pdf

ShowPermissions Plug-in

NOTE: Windows only.

This sample folder includes the windows version executable files of the ShowPermissions sample. ShowPermissions plug-in sample calls the PDDocPermRequest() function to check operation permissions for the front PDF under the current Acrobat product, and saves results to text files.

Now the Acrobat products have different versions (6.x, 7.x) and variations (Reader, Standard, Pro, Approval, ...), and a PDF may have certain level of Reader-Enabled usage-rights, so this plug-in tool can provide useful information about the Acrobat functionality / API availabilities and the PDF permitted operations.

ShowPermissionsSDK7.api here is the binary file of this plug-in developed with Acrobat 7 SDK, and it has been Reader Enabled on Windows. Note that if you directly build the ShowPermissions plug-in sample in the SDK, then your binary output api file won't be able to run in Adobe Reader.

To use the tool, copy this binary file to the Acrobat 7.x or Adobe Reader 7.x plug_ins directory. The plug-in will add an item of "Show Permissions" under Tools menu. When clicked, it will ask the user to select a folder for the output, 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.

ShowPermissionsSDK6.api here is developed with the Acrobat 6 SDK, and it has been Reader Enabled, too. It can be used with Acrobat 6 products which ShowPermissionsSDK7.api cannot work with.

Verify URLs Plug-in

NOTE: Mac OS only.

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.

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 plug-in 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. The sample uses:

Red square = URI associated with link is dead.

Green square = URI associated with link is live.

The annot index is the index into the annot array of the page. The status number is the HTTP status code returned by the server.

Plug-in Wizard

NOTE: Windows only.

Wizard to create plug-ins for Acrobat 7 with Visual Studio .NET 2003

The Plug-in Wizard extends Microsoft Visual Studio .NET 2003 to simplify the creation of Visual Studio C++ projects that create plug-ins for Acrobat 7.

Installation

If Visual Studio .NET is installed when the Acrobat 7 SDK installer is run, the SDK installer will install the necessary files in the Visual Studio .NET installation. If you need to manually install the Plug-in Wizard, copy Acro7PIWiz.ico and Acro7PIWiz.vsz from the SDK installation tree to in the Visual Studio installation tree.

Usage

In Visual Studio .NET, 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 Acro7PIWiz in the Templates pane. Enter a name and location for the project and click OK.

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.

Notes

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.

IAC Samples

Macintosh - Interapplication Communication Samples

AddAnnotations

The AddAnnotations script demonstrates how to create and manipulate the annotation objects that are supported in Acrobat's AppleScript dictionary - text and link annotations.

The script requires a PDF file with at least 3 pages to function properly. 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 **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

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

ObjectProperties

The ObjectProperties script demonstrates how to work with various objects exposed through Acrobat's AppleScript interface including the application, document windows, documents, and pages.

PrintPage

The PrintPage script prompts the user to browse for a PDF file then prints the first page to the default printer.

RotatePages

The RotatePages script 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

The SelectText script demonstrates how to create text selections within a document open in the viewer.

WatermarkJsoAS

The WatermarkJsoAS script demonstrates how to access the Acrobat JavaScript Object to execute addWatermarkFromText() and addWatermarkFromFile() methods. This sample mimics the Windows IAC sample WatermarkJsoVB.

Windows - Interapplication Communication Samples

AdobePDFSilent

This Visual Basic sample demonstrates how to silently print to the Adobe PDF printer. It processes files from a specified input directory (default is the application's current directory) and places output PDF files in a specified output directory (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.

Command Line 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 PDFs from opening an Acrobat viewer, unselect the "View Adobe PDF Results" option in the Adobe PDF print driver printing preferences.

AcrobatActiveXVB

This sample 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 provided a COM-based automation interface (AcroPDF.dll). You can add it to your project's references to have such a new control in your toolbox. Once you drag and drop the ActiveX control to a window in your application, you can open a PDF document window with toolbars. You can also use the APIs provided by the ActiveX control to programmatically manage the window for purposes such as opening a PDF file from a URL or a local disk, page navigation, setting display mode or style, hiding toolbars, etc.

Usage

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

AcroPDFInHTML

This is 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

The user must have an Internet browser, such as 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 to do so in Internet Explorer.

The sample includes four html files. Click the frameset1.htm file to start. A web page with frames will display in the Internet 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 the PDF 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.

ActiveViewVB

ActiveViewVB demonstrates how to use Acrobat's automation interface to display an interactive view of a PDF document within a VB.NET application's 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 and manipulate the view of the PDF.

ActiveViewVC

ActiveViewVC demonstrates how to use Acrobat's 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 and manipulate the view of the PDF.

BasicIacCS

BasicIacCS is a simple sample that provides the minimum code to use Acrobat IAC 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

This simple VB sample demonstrates how to use the Acrobat IAC JavaScript Object in VB 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).

BasicIacOCXCS

BasicIacOCXCS 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 pressing a **Browse** button. The file specified in **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.

BasicIACVB

This is a simple Acrobat IAC VB sample. It includes the code to launch the Acrobat viewer, open a PDF file (`C:\sample.pdf`), and get simple information (number of pages).

BasicIacVC

This simple sample provides the minimum code to use Acrobat IAC in a VC 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. Otherwise, it shows an error message.

DdeOpenVC

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

DistillerCtrlVB demonstrates how to use Acrobat Distiller's 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, e.g.

```
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

DistillerCtrlVC demonstrates how to use Acrobat Distiller's 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, e.g.

```
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

DistillerCtrl_WMVC demonstrates how to use Acrobat Distiller's windows messaging interface to convert PostScript files to PDF in a C++ application.

Usage

The sample allows 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.

As 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

This sample demonstrates how to execute Acrobat JavaScript code by calling the AcroForm OLE ExecuteThisScript method in VB applications. The default code writes information about the active PDF to a new PDF report file, then opens the report. The program provides a dialog for the user to modify, rewrite, and execute their own JavaScript code.

Usage

1. Click **PDF Document** to open or change the active PDF file open in Acrobat.
2. 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.
3. Click **Clear** to clear the text window, then input your JavaScript to execute.
4. Click **Reset** to restore the default JavaScript code.
5. Click **Help** to check the help message.
6. 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 Acrobat JavaScript code from an IAC application.

FillFormCS

FillFormCS demonstrates how to Fill a form from a C# application using IAC.

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

FormsAutomationVB 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.

JSObjectAccessVB

JSObjectAccessVB demonstrates how to access the JavaScript object model through Acrobat's automation interface.

Usage

Before running the sample, the following variables must be configured in JSObjectAccess.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.

JSOFindWordVB

This is a **JSObject** sample that shows how to access Acrobat JavaScript using Visual Basic. The approach is to get the Acrobat 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.

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

RemoteControlAcrobatVC

This is 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

This sample 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.

NOTE: The sort options are no longer valid in Acrobat 6.

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 VB program is located.

StaticViewVB

StaticViewVB 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 through the toolbar and View menu.

StaticViewVC

StaticViewVC demonstrates how to use Acrobat's 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 through the toolbar and View menu.

WatermarkJsoVB

This VB.NET sample 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.

JavaScript Samples

JavaScript samples are divided into seven categories, each in its own folder in the Acrobat 7.x SDK/JavaScriptSupport/Samples folder:

- ADBC
- Inside PDF (Scripts written in JavaScript that "live inside" a PDF file)
- JSCollection
- Multimedia
- Outside PDF (JavaScript files that "live outside" an Acrobat file)
- SOAP
- UI

NOTE: The JavaScript debugger.js file located in the JavaScriptSupport/Debugger folder can be used with Adobe Reader 7.0 to debug JavaScript in a console window. For information on how to install and use this file, please see the "Using the Acrobat JavaScript Debugger" section of the *Acrobat JavaScript Scripting Guide*.

ADBC (Windows-only)

BookList

BookList.pdf is an Acrobat Forms sample using JavaScript ADBC objects to work with a database. The sample provides a graphics interface for you to manage your books. You can retrieve and modify book information, or add or delete a book information record. The data shown in the Acrobat form includes text, number, Boolean, and icon or image.

This sample demonstrates how to use the Acrobat-specific JavaScript ADBC object to establish a connection with an ODBC database, and how data is retrieved from, manipulated within, and written to a database using standard SQL queries. The sample also shows how to use various form fields as a GUI. All functionality is created by JavaScript code that is used in the Doc level, field actions, and page actions.

To run the sample, you need to define an ODBC data source called **MyBookListSample** with the sample database file booklistdb.mdb. For example, here are the steps on Windows 2000:

1. Select **Start > Settings > Control Panel > Administrative Tools > Data Sources (ODBC)**.
2. Choose the System DSN tab. Click **Add...**
3. In the **Create New Data Source** window, select **Microsoft Access Driver (*.mdb)**; click **Finish**.
4. Enter the data source name : "MyBookListSample". Click **Select**.

5. Browse to the appropriate folder (e.g., under Windows browse to Adobe 7.x SDK) and select the sample data source `booklistdb.mdb`.
6. Click **OK** to finish.

After you have completed the data source setup, loading the sample file **BookList.pdf** into Acrobat will automatically establish the connection to the ODBC database. The data tables, item ID list, and information for a book should be displayed. Then you can use the following buttons to work with the database:

Connect - Initialize the connection or reset the connection to the database.

New - Clear the form and fill in data for a new record. When finished, you must click Save to add the record to the database.

Save - Update an existing record or add a new record in the database.

Delete - Delete an item from the database.

Close - Close the database and quit the PDF form.

Brief instructions and confirmation messages are shown in the bottom text field.

In addition to the text data, you can specify an image file to be displayed for any entry. To add or re-specify such an image file, click the big button titled **Book Image**, then select a PDF file (or image file of a type supported by Acrobat). The image will be displayed as the new face of the button. Note that the sample database records only the filename, so you must put all new image files in the same directory as the sample PDF file. If the image file is not found, a default image will display.

This is a sample only, and some GUI details may not be perfectly implemented. Further improvements will need to be made as you use the code to develop your own applications.

Inside PDF

ConvertTime

`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. Press 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.

EmbeddingFormData

The `EmbeddingFormData.pdf` sample 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, 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.

JavaScriptSnippet1

The sample SDKJSSnippets1.pdf document 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 Acrobat 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 - uses a new Acrobat 7 method to create a popup menu.
- Metadata - shows a quick way to get document metadata in XML-formatted text.
- Full screen - displays a PDF 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.

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.

OCGApiSample

OCGApiSample.pdf is a sample to exercise JavaScript APIs for PDF Optional Content Groups. The code is embedded as document level JavaScript and executed through a JSADM dialog.

Usage

To run the sample, first open the JavaScript console as well as the Layer panel, then press the button in the PDF to pop up a dialog. There are eight buttons for you to try. Check the results on the Layer panel, document window, and JavaScript console. Do not save the changes unless you want to.

JSCollection

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

Multimedia

EventState

This Acrobat multimedia sample demonstrates two ways for event listeners to have a local persistent state. It is particularly of help 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 JavaScripts. 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 and higher. 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.

ScriptEvents

This Acrobat multimedia sample demonstrates JavaScript commands sent from Windows Media, 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 works on Mac, too.

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 bottom 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 and higher. 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.

Trophy

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 and higher. 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 installed.

TwoPartInvention

This is a PDF with sheet music you can click to playback. 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 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 and higher. 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.

Outside PDF

These are JavaScript (.js) files that "live" outside an Acrobat file -- for example, the code used in JavaScript folders, the console, and batch processes. In order to distinguish the SDK sample js files from Acrobat build in files, "sdk" was added in front of the js file name in each sample. To use a sample js file, copy it to the JavaScript folder under Acrobat, or to the user's JavaScript folder.

AddSignature

This sample 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 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, 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 PDFs 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 DPath (e.g., `/C/DrTest.pfx`). Its password is "testpassword".

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

Beginning with Acrobat 7.0, 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 SDK JavaScript Reference document.

AnnotSample

AnnotSample is a folder level JavaScript code to exercise the annotation APIs useful in reviewing workflow. It can be used with Rights-Enabled PDFs in Reader as well as regular PDFs in Acrobat.

Usage

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

- Set annotations 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.

To run the sample, you need to specify a file path in the dialog for a data repository in your environment. The file path must be a safe path and in the 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.

GoToBookmark

The `GoToBookmark` sample provides a utility tool for users to get to a bookmark in a PDF document in Acrobat. It adds a menu item to search for a specific bookmark in a PDF

document. 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 the viewer you should see a new item **Go To Bookmark**. It is located under the **Advanced** menu in the Acrobat viewers and located under the **Tools** menu in Reader. Click it to get a dialog box to fill in your search.

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 **Advanced > Go to Bookmark...**, enter the string `"outside pdf"`, 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 with a large number of bookmarks. To cancel the process, press **Esc** on Windows or **Command-period** on Macintosh. A progress bar is implemented using the thermometer JavaScript object.

Presentation Monitor

This sample 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 to enter the number of minutes you plan to use for the presentation. After that the monitor shown in the top of 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 navigated.
- When the mouse enters/exits a page icon, the page image will be shown/hidden in the top left corner.
- Click a page icon and you can go to that page.
- A check box (the second one in the top-right corner): check/uncheck to toggle 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.

PresentationNote

This sample 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. Enter the number of minutes before the start of the presentation and click the **OK** button to close the dialog. 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.

SilentPrint

`SilentPrint` is a folder level JavaScript code to exercise the print APIs. It can be used in Reader as well as Acrobat. There are many print options that users may set up in the JavaScript print parameters. See Acrobat JavaScript reference and guide SDK documents for further information about silent print functionality.

Usage

The sample will add "Silent Print" under the File menu. Click it to print the front document to the default printer without user interface.

TextExtract

`sdkTextExtract.js` is a folder level JavaScript sample that demonstrates how to extract the text in PDF page contents and save it to a local file. JavaScript `Doc.getPageNthWord` method is used to get the words one by one from the current page displayed, 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 save the text extracted from the current page. You can use MS Word to view the text file.

SOAP

Eliza

The Eliza.pdf sample provides an Acrobat Forms front end to a psychoanalyst 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 infamous Eliza computer psychologist from a sepia-tainted time.

GetStockPrice

This GetStockPrice.pdf sample 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.

SOAPCollabSample

This folder level JavaScript sample 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.

Please see the comments in the file for more information. In addition, the document *Acrobat Online Collaboration: Setup and Administration*, available from <http://partners.adobe.com>, 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. Please go to <http://www.ensemble-systems.com/pdfsamples/index.html> for more information.

To use the sample, copy `sdkSOAPCollabSample.js` into your Acrobat JavaScript folder. Open the **Edit > Preferences > Reviewing** panel, where you will see a new collaboration server type. Select it and input the Server Settings as: <http://m43.ensemsys.com/jboss-net/services/PDFCollaboration?wsdl>. Then open a PDF in your Internet browser to perform an online review. When the PDF 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 is closed, the updated annotation data is automatically sent to the server.

UI

AddToolButton

This folder level JavaScript sample, `AddToolButton.js`, demonstrates how to add a toolbar button to Acrobat's toolbar 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 pop up a JSADM dialog box to fill in. You must input a device-independent path to specify a PDF or image file (e.g. JPEG file) as 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. These image files were created by Adobe Acrobat SDK team.

ADMCollection

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

Usage

Open this PDF sample file and press the buttons to pop up the JSADM dialogs. Fill in text values or select items as instructed, and check the feedback in the JavaScript console window. To learn about the implementation, you can look at the JavaScript code inside the PDF through the Acrobat Advanced menu.

ToolbarButton

This sample is a PDF document demonstrating how to add and remove toolbar button icons 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 tool button to Acrobat toolbars.

Usage

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

Plug-in Samples

The plug-in samples in the SDK are listed and described below.

Some plug-ins install a menu item on the **Acrobat SDK** submenu. In Acrobat Standard and Acrobat Professional, the **Acrobat SDK** submenu is located on the **Advanced** menu. In Adobe Reader, the **Acrobat SDK** submenu is located on the **Tools** menu.

ActionHandler

ActionHandler demonstrates how to create a new PDAAction type. PDAActions can be tied to various events within the viewer e.g. the mouse up over link annotations, page open

events, form field triggers, etc. 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, 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 on 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, for presenting an edit properties dialog, and for creating the action dictionary in the PDF. The routine RegisterActionHandler is called during plug-in initialization. It initializes the callback structure and then registers it using:

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

with the parameters being 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 then a text string for the popup list item.

BasicPlugin

This sample plug-in provides the minimum code required to enable developers to get started. The code adds a new menu item under the **Acrobat SDK** submenu and displays a simple message. Recall that in the Acrobat viewers, the **Acrobat SDK** submenu is installed on the **Advanced** menu; in 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

BatchCommand 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:

1. Click **Advanced > Batch Processing... > Edit Batch Sequences** and create a new batch sequence.

2. Click **Select Commands** and add the "Isolate Form Data" and "Lock Form Fields" commands to the sequence.
3. 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.

CivilDiscovery

This sample demonstrates an automated workflow for the production phase of discovery in support of civil litigation. In this phase, certain pages 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. Adds a header to all the pages in the new PDF file.
3. Adds a footer to all the pages in the new PDF file.
4. Adds a "BatesNo" key with its corresponding value to the Info dictionary of the newly created PDF file.
5. Saves the newly-created pdf file with the name and at the directory specified.

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

Usage

To run the sample on Windows, copy the files_win.csv, test.pdf and test1.pdf to C:\. On the Mac OS, copy the files_mac.csv, test.pdf and test1.pdf, similarly, 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 will process 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.

DdeServer (Windows Only)

DdeServer 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.

DebugWindowADM

Demonstrates how a plug-in can export its own HFT (Host Function Table). An HFT is a useful mechanism for exposing common functionality that can be used by any plug-in.

Functionally, the plug-in provides a window that can be written to for the purpose of capturing debug statement output.

The visibility of the output window is toggled through the **Acrobat SDK** submenu's **Show/Hide Debug Window** menu item. Many of the samples supplied with the SDK print output to the window. Take a look at "ActionHandler", "Stamper", or "RplcFileSystem" to see how to import and make use of the HFT.

This plug-in uses the Adobe Dialog Manager (ADM) to display the output window. For Windows and the Mac OS, ADM provides a platform-independent interface for displaying and controlling the output window.

DocSign

This plug-in sample 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.

Usage

To use DocSign, change the default signature handler in Acrobat Preferences to either Ask when signing or to the DocSign handler. By default, Acrobat uses a pre-selected standard signature handler. After that, DocSign is used through the same user interface points as the standard signature handler. That is, the Sign menu bar item, the Document > Digital Signatures > Sign this document menu item, and the File > Save as Certified Document menu item.

Limitations

This sample DOES NOT provide true digital signatures. It provides a skeleton of a PubSec API-based plug-in to which a developer MUST add their own encryption routines.

ExternalView

Simple application that exercises the IAC interface exposed by the External Window 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 the application.

ExternalWindow

ExternalWindow 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.

PDFBinder

This sample shows how to grammatically 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 may be running without any graphical user interface on the screen, so it can be executed conveniently from other programs using C IAC, VB IAC, or JavaScript to meet your enterprise workflow needs.

Usage

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

1. Click the menu item to run the program. This method uses a fixed file location that is hard coded (C:\test\PDFBinder on Win, MacHD:test:PDFBinder on Mac).
2. Press down 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, VB IAC, or JavaScript.

Limitations

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`.

About the test files

Most of the test files in the project's Testfiles folder are originally from the book of "Acrobat 5: classroom in a book" with the copyright by Adobe Systems Incorporated and all rights 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.jpg, and giffile.gif, were made by Adobe Acrobat SDK team.

RplcFileSystem

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. The output window (DebugWindowHFT or console) is a good way to see how Acrobat interacts with the file system implementation.

Usage

The replacement file system can be exercised by selecting the **Replacement File System** menu item on the **Acrobat SDK** submenu. On Windows and the Mac OS, the plug-in makes use of the DebugWindowHFT so you can see every call to file system's callbacks in the output window.

On UNIX, where ADM APIs are not available, launch the viewer via terminal (console) rather than the GUI in order to display the file system callbacks in the console window. If the viewer is launched via the GUI, then there is no output window.

NOTE: The volume of messages sent to the DebugWindow can be quite high; as such, Acrobat can become very sluggish, especially on the Macintosh.

Implementation Details

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

SecurityHandler

SecurityHandler 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 > Document Security...** menu item, "SDK Security Handler" appears as one of the options in the Security Options dropdown list. Clicking **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

SelectionServer demonstrates how to implement a minimal selection server.

Usage

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

ShowPermissionsSDK7

This sample 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 select. The output's filenames are the PDF filename plus -permissions.txt or .xls. The tab delimited file can be opened by Excel.

Now the Acrobat products have different versions (6.x, 7.x) and variations (Reader, Standard, Pro, Approval, ...), and a PDF may have 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.

Snippet Runner

The SnippetRunner plug-in provides infrastructure and utility functions to support execution of Acrobat plug-in code snippets. A snippet is a small complete portion of Acrobat plug-in code.

More than 100 sample code snippets are provided in the SnippetRunner project to demonstrate Acrobat API methods. However, the SnippetRunner plug-in also allows developers to quickly prototype Acrobat API calls without the overhead of writing and verifying a complete plug-in.

For complete information and details regarding the SnippetRunner, its environment, user interface and how to write and load original snippets, see the SnippetRunner Cookbook document, **SnippetRunnerCookbook.pdf**, provided in the "Intro to SDK documents" directory and in the SnippetRunner's Example Files directory.

Introduction to the Common User Interface

Beginning in Acrobat SDK 7.0.5, PDFL SDK 7.0.5, and Linux Reader version 7.0, SnippetRunner interaction is via a common Java-based graphical user interface rather than the previous separate interfaces, such as the ADM interface (Acrobat Win/Mac) or the command line interface (PDFL and Acrobat/UNIX).

This Common User Interface acts as a client to its associated SnippetRunner back-end -- the SnippetRunnerServer Acrobat plug-in or SnippetRunner PDFL application -- and provides Acrobat and PDFL developers with a single cross-platform GUI. The Common Interface executes snippet commands and provides rich information about snippets, output and document status.

Snippets that require support

Some snippets will require external "files". These can either be sample files for input, or resources (for UI artifacts).

Example Files

Example files are delivered within the `Examples` directory of the SnippetRunner plug-in. When executing any snippet for the first time, the user is prompted for the root of the `SnippetRunner` folder. This persists between application instantiations, and utility functions exist that allow users to open documents in the examples folders by purely specifying the file name (no path required).

The following snippets are provided with the Acrobat 7 SDK. See the associated snippet documentation for details (accessible from the SnippetRunner dialog that becomes available when the plug-in loads):

- ACEEnumProfilesSnip.cpp
- ACEEnumSettingsSnip.cpp
- ACEGetWorkingSpaceSnip.cpp
- ACETransPDETextColorSnip.cpp
- ADMBuildInDialogsSnip.cpp
- ADMDialogUserdataSnip.cpp
- ADMFloatingDialogSnip.cpp
- ADMHierarchyListSnip.cpp

- ADMListDlgSnip.cpp
- ADMModalDialogSnip.cpp
- ADMPromptDlgSnip.cpp
- ADMVersionSnip.cpp
- ASBigFileSnip.cpp
- ASCabPutGetSnip.cpp
- ASChangeTempFileSysSnip.cpp
- ASDateSnip.cpp
- ASFileIteratorSnip.cpp
- ASGetConfigurationSnip.cpp
- AVAlertSnip.cpp
- AVAppFrontDocChangeNotSnip.cpp
- AVAppPrefsSnip.cpp
- AVAppRegisterForContextMenuSnip.cpp
- AVAppRegisterForPageViewDrawingSnip.cpp
- AVDocAVWindowDidChangeSnip.cpp
- AVDocCloseNotSnip.cpp
- AVDocGetSelectionTypeSnip.cpp
- AVDocOpenNotSnip.cpp
- AVDocShowAnnotPropertiesSnip.cpp
- AVDocWindowWasAddedSnip.cpp
- AVDocWindowWasRemovedSnip.cpp
- AVEnumActionHandlerSnip.cpp
- AVPageViewDrawRectSnip.cpp
- AVPageViewToggleWireframeDrawingSnip.cpp
- AVPrintSnip.cpp
- AVSaveAsRtfSnip.cpp
- AVWindowMaximizeCurrentPageViewSnip.cpp
- AboutAcrobatSDKSnippets.cpp
- AddImageMetadataSnip.cpp
- AddImageSnip.cpp
- AddPageMetadataSnip.cpp
- AddStructureSnip.cpp
- AddTagSnip.cpp

- AddTextSnip.cpp
- AddXObjectStructureSnip.cpp
- CallJsCreateButtonSnip.cpp
- CallJsGetFieldValueSnip.cpp
- CallJsResponseSnip.cpp
- CallJsSignatureFieldSnip.cpp
- CallJsTextFieldSnip.cpp
- ClassMapSnip.cpp
- CloseFrontDocSnip.cpp
- ColorSelectedBookmarksSnip.cpp
- ColorSetupSnip.cpp
- ConvertFlate2JPXSnip.cpp
- ConvertOCGsToRadioButSnip.cpp
- CosCryptGetVersionSnip.cpp
- CosDecryptDataSnip.cpp
- CosDictKeyNameStringSnip.cpp
- CosDoc64Snip.cpp
- CosEncryptDataSnip.cpp
- CosNumberObjRangeSnip.cpp
- CosObjCompressionSnip.cpp
- CosObjDecompressionSnip.cpp
- CosObjWeakReferenceSnip.cpp
- CosObjectExplorerSnip.cpp
- CosStream64Snip.cpp
- CreateAnnotOCsSnip.cpp
- CreateContentXORSnip.cpp
- CreateDocStructSnip.cpp
- CreateImageContentOCsSnip.cpp
- CreateTextContentOCsSnip.cpp
- DocMetadataSnip.cpp
- EmitPostScriptSnip.cpp
- EnumAVConvExtSnip.cpp
- ExploreMetadataSnip.cpp
- ExploreStructSnip.cpp

- ExportFormDataSnip.cpp
- FontInfoSnip.cpp
- FormCalculationsSnip.cpp
- FullScreenTransitionsSnip.cpp
- GetDocKeywordSnip.cpp
- GetDocMetadataSnip.cpp
- IdleProcSnip.cpp
- ImageInfoSnip.cpp
- InvokeAccessibilityCheckerCmdSnip.cpp
- InvokeCreateAllThumbsCmdSnip.cpp
- InvokeDeleteAllThumbsCmdSnip.cpp
- InvokeDeleteCmdSnip.cpp
- InvokeFlattenOCGsCmdSnip.cpp
- InvokeMakeAccessibleCmdSnip.cpp
- InvokeOpenOptionsCmdSnip.cpp
- InvokeSummarizeCmdSnip.cpp
- JPXColorSpaceExplorerSnip.cpp
- JPXPaletteExplorerSnip.cpp
- LoadHowToSnip.cpp
- MakeBookmarkSnip.cpp
- OCActionControlSnip.cpp
- OCGUIReorderSnip.cpp
- OCTextAutoStateSnip.cpp
- ObjShiftSnip.cpp
- OptContNotificationTracerSnip.cpp
- PDCreateMasterOCGSnip.cpp
- PDDocDidDeletePagesNotSnip.cpp
- PDEContentExplorerSnip.cpp
- PDEPathDrawCurveSnip.cpp
- PDEPathDrawLineSnip.cpp
- PDEPathDrawRectSnip.cpp
- PDEPathExplorerSnip.cpp
- PDEPathToggleVisibilitySnip.cpp
- PDOCConfigCreateSnip.cpp

- PDOCConfigExplorerSnip.cpp
- PDOCGChangeLockedStateSnip.cpp
- PDOCGExplorerSnip.cpp
- PDOCGToggleIntentSnip.cpp
- PDOCSetDefaultConfigSnip.cpp
- PDPPageNotifyContentsChangeSnip.cpp
- PDPPageSetTransparencySnip.cpp
- RaiseExcepSnip.cpp
- RegisterFileConverterSnip.cpp
- RemoveEmbeddedFontSnip.cpp
- ReplaceMethodSnip.cpp
- ResetFormSnip.cpp
- RoleMapSnip.cpp
- SecureDocumentSnip.cpp
- SeparationsPreviewSnip.cpp
- SetDocBaseURLSnip.cpp
- ShowTextFieldNamesSnip.cpp
- SimpleSnip.cpp
- SmartPDPPageSnip.cpp
- SnapZoomSnip.cpp
- TextChangeColourSnip.cpp
- TextExtractionSnip.cpp
- TextInfoSnip.cpp
- TransHandlerSnip.cpp
- TransformMetadataSnip.cpp
- UserPropertiesExplorerSnip.cpp

Stamper

Stamper 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** submenu's **Stamper Annotations** menu item.

Usage

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

Starter

Starter is 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

Transparency 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\testfiles** folder to test the functionality of the plug-in.

UncompressPDF

UncompressPDF is 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.

WeblinkDemo

WeblinkDemo 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 send information to Netscape 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 debug window indicating that it has been called.

NOTE: In order for messages to appear in the debug window, Web links must be opened in your Web browser, as indicated by your preferences in **Edit > Preferences > WebCapture**. If your preference is set to open Web links in Acrobat, when you click on a link, the messages will not appear.

WordFinder

WordFinder 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 filename 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 dialogin which to input a word offset; on UNIX (no ADM APIs), the offset is hardcoded into the plug-in source and no dialog is presented. The default value is set as 0 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.

Other Samples

XML Samples

SaveAsXML Samples

The SaveAsXML samples are used in conjunction with the SaveAsXML plug-in, which is documented in a SDK document, SaveAsXml.pdf, with title of *"Information For Developers Using The SaveAsXML Plug-in"*. SaveAsXML is a plug-in for Adobe Acrobat which extends the "Save as type" choices in the SaveAs dialog 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 Acrobat package. The SDK SaveAsXML samples are sample files that demonstrate how to create such customer Mapping Tables.

As Mapping Tables, every sample has a Root note, and it defines the output file format, e.g. HTML-3-20, menu name to be shown in Acrobat SaveAs dialog, e.g. HTML 3.2 (*.htm, *.html), and other internal information. Then it is followed by various directives with hierarchical data, which provide the instructions how to convert PDF elements and data into XML objects when the SaveAsXML processor walks through the structure of a Tagged PDF document. Besides, there is a DTD (Document Type Definition) file provided in SDK. It is to define the legal building blocks of an XML document. The SaveAsXML processor requires that the Mapping Tables must be valid in accordance with this DTD.

Developers can take the samples as templates to create their own Mapping Tables. The sample files were created using FrameMaker+SGML 6.0, but any XML (or SGML) editor can be used to edit the .xml versions of the Mapping Tables.

To use a SaveAsXML sample, copy it to the SaveAsXML folder under Acrobat, for example, C:\Files 7.0, then you can use the new type added to the Acrobat SaveAs dialog.

There are two groups of samples: introductory and advanced. The advanced samples are found in the `SaveAsXML /Advanced` directory. The introductory samples are found on the first level of the `SaveAsXML` directory.

Introductory Samples

The following samples are found on the first level of the `SaveAsXML` directory. Developers can use these samples as starting points to create richer extraction tables for their own needs.

- `SDKBookmarkExtraction` - Shows how to extract the bookmarks from a document.
- `SDKImageExtraction` - Shows how to extract the images from a document.
- `SDKTextExtraction` - Shows how to extract the text from a document.
- `SDKDocInfoExtraction` - Shows how to extract the document info from a document.

In addition, `MappingTable.dtd` provides the DTD to which mapping tables must adhere.

Advanced Samples

The samples found in the `Advanced` folder are more solution-centric. Developers can either use these as is, or refine them to suit their applications. Most developers will use them as templates only. They provide starting points for real-life situations. For example, if a developer wishes to provide a mapping table that performs comprehensive XML-based output (to some arbitrary DTD), the `XML-1-00-with-all-attrs.xml` sample would make a good template. The title of each sample file provides an idea of what each one does, and the following is a list of the samples:

- `HTML-3-20-Access.xml` - Converts PDF documents into HTML 3.2 with Access.

- `HTML-3-20-CSS-1-00.xml` - Converts PDF documents into HTML 3.2 with CSS 1.0. CSS(Cascading Style Sheets) is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents
- `HTML-3-20-CSS-2-00.xml` - Converts PDF documents into HTML 3.2 with CSS 2.0.
- `OEB-1-00-doc.xml` - Converts PDF documents into OEB 1.0. OEB(Open eBook) is a dialect of XML and allows users to create their own tags while remaining good compatible with HTML.
- `OEB-1-00-doc-paged.xml` - Converts PDF documents with page data into OEB 1.0.
- `XHTML-1-00-CSS-1-00.xml` - Converts PDF documents into XHTML 1.0 with CSS 1.0.
- `XHTML-1-00-CSS-2-00.xml` - Converts PDF documents into XHTML 1.0 with CSS 2.0.
- `XML-1-00-Full-Info.xml` - Converts PDF documents with information data into XML 1.0.
- `XML-1-00-with-all-attrs.xml` - Converts PDF documents with attributes into XML 1.0.







