Developing for Adobe® Reader®

Adobe® Acrobat® SDK

November 2006

Version 8.0

Adobe® Acrobat® SDK 8.0 Developing for Adobe Reader for Microsoft® Windows®, Mac OS®, Linux®, and UNIX®

Edition 1.0, November 2006

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

Contents

Preface

Adobe® Reader® belongs to the Adobe Acrobat® family of products, and is used for viewing, navigating, and printing PDF documents. For more information on the Acrobat family of products, see http://www.adobe.com/go/acrobat_developer.

What's in this guide?

This guide provides an introduction to those portions of the Adobe Acrobat Software Development Kit (SDK) that pertain to your development efforts for Adobe Reader. It provides a general overview of the types of things you can do with the SDK and the technologies that are available to you through the SDK. This document provides a starting point for developers who would like to understand how to extend or customize Adobe Reader, and provides information clarifying how such efforts differ from those intended for Acrobat.

It is possible to extend and customize Adobe Reader by using the Adobe Acrobat SDK to write JavaScript™ code, implement interapplication communication, and write plug-ins. This document indicates the relevant subsets of the JavaScript APIs, interapplication communication APIs, and the Acrobat and PDF Library APIs. It describes each of those collections of APIs and their intended purposes, and clarifies what is available on all supported platforms.

Who should read this guide?

This guide is meant for developers who are either new to Adobe Reader development or have experience with the Acrobat SDK.

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 roadmap containing descriptions of all the documentation in the Acrobat SDK.	*Acrobat SDK Documentation Roadmap*
A description of known issues and implementation details specific to the various platforms supported in the Acrobat SDK.	*Readme*
Answers to frequently asked questions about the Acrobat 8.0 SDK.	*Developer FAQ*
The new features in this SDK release.	*What's New*

For information about	See
A general overview of the capabilities and usage of the Acrobat SDK.	*Overview*
An introduction to those portions of the Acrobat SDK that pertain to development efforts for Adobe Reader.	*Developing for Adobe Reader*
Descriptions and implementation details for samples included with the Acrobat SDK.	*Guide to SDK Samples*
An overview of the SnippetRunner tool and the plug-in snippets provided with the Acrobat SDK.	*Snippet Runner Cookbook*
A description of how to develop external applications that use Apple events, AppleScript, DDE, or OLE to control Acrobat or Adobe Reader or render PDF documents.	*Developing Applications using Interapplication Communication*
Detailed descriptions of the APIs available for Apple events, AppleScript, DDE, or OLE to control Acrobat or Adobe Reader or render PDF documents.	*Interapplication Communication API Reference*
An overview of how to use JavaScript to develop and enhance standard workflows in Acrobat or Adobe Reader.	*Developing Acrobat Applications using JavaScript*
Detailed descriptions of the JavaScript APIs for developing and enhancing standard workflows in Acrobat or Reader.	*JavaScript for Acrobat API Reference*
A detailed description of the PDF file format.	*PDF Reference*
A description of how to develop plug-ins for Acrobat and Adobe Reader, as well as PDF Library applications.	*Developing Plug-ins and Applications*
A detailed description of the APIs available to create plug-ins for Acrobat and Adobe Reader, as well as PDF Library applications.	*Acrobat and PDF Library API Reference*
Describes the syntax and semantics of the PostScript language and the Adobe imaging model.	*PostScript Language Reference, third edition*

1 Introduction

This chapter describes the supported platforms for development using the Acrobat SDK, and summarizes the technologies available within the Acrobat SDK.

Supported platforms

This section describes the requirements for using the interapplication communication and Acrobat core and extended APIs on all supported platforms. The following platforms are currently supported for development with Adobe Reader:

- Windows®
- Mac OS
- Linux®
- Solaris®
- AIX®
- HP-UX

Details for each platform are described below.

Windows

Versions

- Microsoft Windows 2000 with Service Pack 2
- Microsoft Windows XP Professional or Home Edition

Development environments

- Microsoft Visual Studio .NET 2003
- Microsoft Visual Studio 2005

Note: For more information on the Windows environment, see the *Overview* and *Developing Plug-ins and Applications*.

Mac OS

Versions

- Mac OS X versions 10.2.8 or later.

Development environment

- Xcode 2.3

Note: For more information on Mac OS development environments, see the *Overview* and *Developing Plug-ins and Applications*.

Linux

Machine requirements

- 32-bit Intel® Pentium®-class processor, 128 MB RAM (256 MB recommended), 70 MB hard disk space.

Distributions and versions

- Red Hat® Enterprise Linux AS version 3.0, with Linux kernel version 2.4.21
- Red Hat Enterprise Linux AS version 4.0, with Linux kernel version 2.6
- Red Hat Enterprise Linux ES version 3.0, with Linux kernel version 2.4.21
- Red Hat Enterprise Linux ES version 4.0, with Linux kernel version 2.6
- Red Hat Linux Desktop
- Red Hat Linux version 9.0, with Linux kernel versions 2.4.20 through 2.6
- SuSE Linux Enterprise Server version 9, with Linux kernel version 2.6
- SuSE Linux Professional version 9.2, with Linux Kernel kernel 2.6.4
- Turbolinux 10 Desktop, with Linux kernel version 2.6
- Red Flag Linux Desktop 4.0

Note: Not all versions of Red Hat Linux will automatically install the GNOME GTK+ Library, which is required for developing plug-ins for Adobe Reader on Linux. To ensure that your Adobe Reader plug-ins will compile, make sure the GNOME GTK+ Library has been installed on your system.

Supported browsers

- Mozilla versions 1.73 and 1.8
- Netscape version 7
- Firefox version 1.0

Development environment

- Standard GNU Compiler: gcc version 3.2

Solaris

Machine requirements

- UltraSPARC® or UltraSPARC IIIi processor, 128 MB RAM, 70 MB hard disk space.

Versions

- Solaris Operating System versions 8 and 9

Supported browsers

- Mozilla version 1.73
- Netscape version 7

Development environment

- Standard GNU Compiler: gcc version 3.2

AIX

Machine requirements

- RISC System/6000® or IBM Power5 processor, 128 MB RAM, 70 MB hard disk space.

Versions

- IBM AIX versions 5.2 and 5.2.0.35

Supported browser

- Mozilla version 1.73

Development environment

- Native compiler: xIC version 6.0

HP-UX

Machine requirements

- 32 bit PA-8000x processor, 128 MB RAM, 70 MB hard disk space.

Versions

- HP-UX versions 11 and 11i

Supported browser

- Mozilla version 1.6

Development environment

- HP ANSI C++ native compiler: aCC version A.03.33

Technologies available within the Acrobat SDK

The primary technologies for creating software to extend or customize Adobe Reader are JavaScript, interapplication communication, and plug-ins. For information about choosing an appropriate technology for your project, see the *Overview* guide.

It is important to consider the role of Adobe LiveCycle® Reader Extensions in your development efforts with JavaScript and plug-ins. Though the APIs available for Adobe Reader are normally limited in both cases, additional APIs can be used for a given PDF document if that document is rights-enabled, meaning that it has additional usage rights. LiveCycle Reader Extensions is a server product that enables document producers and creators to quickly and easily embed additional usage rights into PDF documents, which results in extra functionality when the documents are opened. The extra functionality makes the following activities possible:

- Saving forms with results offline
- Connecting forms to databases or online services
- Attaching files and media clips
- Saving copies of documents with changes intact

- Submitting completed documents electronically
- Digitally signing documents
- Sharing documents with others to review and add comments using intuitive markup tools such as electronic sticky notes, highlights, and text strike-throughs

Note: With LiveCycle Reader Extensions, it is not necessary to distribute any plug-ins or other special software to implement these features.

For more information on the Adobe LiveCycle products, see the *Developer FAQ*.

JavaScript

JavaScript is a platform-independent scripting language with which you can customize the behavior of PDF documents in Acrobat or Adobe Reader, as well as the behavior of Acrobat or Adobe Reader itself. You will find that using JavaScript is, in many cases, much easier than writing plug-ins.

Note: Adobe Reader support for JavaScript is limited. For details, see "JavaScript APIs" on page 10.

Interapplication communication

Acrobat and Adobe Reader provide support for interapplication communication (IAC) through OLE automation and DDE on Windows platforms, and through Apple events and AppleScript on Mac OS. IAC is only supported on Windows and Mac OS platforms, and is not supported on Linux or UNIX® platforms.

Note: Adobe Reader support for IAC is limited. For details, see "Interapplication Communication" on page 25.

Plug-ins

Plug-ins are dynamically linked extensions to Acrobat or Adobe Reader, and can be developed on all supported platforms. A plug-in can extend or customize the functionality of Acrobat or Adobe Reader, and can be integrated into the user interface. Plug-ins are written in ANSI C/C++ using the Acrobat APIs.

In order to write a plug-in for Adobe Reader, you must create a Reader-enabled plug-in.

Note: Adobe Reader support for the Acrobat core and extended APIs is limited. For details, see "Plug-ins" on page 28.

2 JavaScript APIs

With Adobe Reader, JavaScript can be used for a number of tasks:

- To develop and process Acrobat forms and XML forms
- To customize the behavior and appearance of a PDF document
- To facilitate online team review
- To implement security policies
- To interact with web services
- To customize the behavior and appearance of Adobe Reader itself

Note: The JavaScript debugger available in Acrobat is not normally available in Adobe Reader, though debug messages can be triggered to appear in the console. The complete debugger functionality can be enabled in Adobe Reader on Windows and Mac OS platforms. For details, see *Developing Acrobat Applications using JavaScript*.

As you learned in "Technologies available within the Acrobat SDK" on page 8, additional usage rights may be applied to a PDF document using LiveCycle Reader Extensions. For detailed information on which JavaScript APIs are available in rights-enabled PDF documents, see *Developing Acrobat Applications using JavaScript*.

Objects, properties and methods

On all supported platforms, JavaScript can be used for processing within a single document, processing for a given page within a document, and processing for a given form field.

The following table JavaScript objects, properties, and methods available in Adobe Reader contains a list of the JavaScript objects, properties, and methods that can be used with Adobe Reader.

Note: Some of the objects listed below, such as those related to the console, debugger, media players, and text-to-speech, are not available for all platforms. Also, many properties and methods are only available within certain contexts and circumstances. For details, see the *JavaScript for Acrobat API Reference* and *Developing Acrobat Applications using JavaScript*.

JavaScript objects, properties, and methods available in Adobe Reader

Object	Properties	Methods
`Alerter`		`dispatch`
`Alternate-Presentation`	`active` `type`	`start` `stop`

`Annotation`	`alignment`
	`AP`
	`arrowBegin`
	`arrowEnd`
	`attachIcon`
		`destroy`
		`getProps`
		`getStateInModel`
		`setProps`
		`transitionToState`

Object	Properties	Methods
Annotation (Continued)	author borderEffectIntensity borderEffectStyle callout caretSymbol contents creationDate dash delay doc doCaption fillColor gestures hidden inReplyTo intent leaderExtend leaderLength lineEnding lock modDate name noteIcon noView opacity page point points popupOpen popupRect print quads rect readOnly refType richContents richDefaults rotate seqNum soundIcon state stateModel strokeColor style subject textFont textSize toggleNoView type vertices width	

Object	Properties	Methods
Annot3D	activated context3D innerRect name page rect	
app	activeDocs calculate constants focusRect formsVersion fromPDFConverters fs fullscreen language media monitors numPlugIns openInPlace platform plugIns printerNames runtimeHighlight runtimeHighlightColor thermometer toolbar toolbarHorizontal toolbarVertical viewerType viewerVariation viewerVersion	addMenuItem addSubMenu addToolButton alert beep beginPriv browseForDoc clearInterval clearTimeOut endPriv execDialog execMenuItem getNthPlugInName getPath goBack goForward hideMenuItem hideToolbarButton launchURL listMenuItems listToolbarButtons openDoc popUpMenu popUpMenuEx removeToolButton response setInterval setTimeOut trustedFunction trustPropagatorFunction
app.media	align canResize closeReason defaultVisible ifOffScreen layout monitorType openCode over pageEventNames raiseCode raiseSystem renditionType	addStockEvents alertFileNotFound alertSelectFailed argsDWIM canPlayOrAlert computeFloatWinRect constrainRectToScreen createPlayer getAltTextData getAltTextSettings getAnnotStockEvents getAnnotTraceEvents getPlayers

Object	Properties	Methods
app.media (Continued)	status trace version windowType	getPlayerStockEvents getPlayerTraceEvents getRenditionSettings getURLData getURLSettings getWindowBorderSize openPlayer removeStockEvents startPlayer
Bookmark	children doc parent	execute
Certificate	binary issuerDN keyUsage MD5Hash SHA1Hash serialNumber subjectCN subjectDN ubRights usage	
Collab		addStateModel documentToStream removeStateModel
color	transparent black white red green blue cyan magenta yellow dkGray gray ltGray	convert equal
Column	columnNum name type typeName value	
ColumnInfo	name description type typeName	

Object	Properties	Methods
console		clear hide println show **Note:** Only println is supported on Linux and UNIX platforms.
Data	creationDate description MIMEType modDate name path size	
Dialog		enable end load store
Directory	info	connect
DirConnection	canList canDoCustomSearch canDoCustomUISearh canDoStandardSearch groups name uiName	search
Document	alternatePresentations author baseURL bookmarkRoot calculate creationDate creator dataObjects delay disclosed docID documentFileName dynamicXFAForm external fileSize hidden hostContainer icons info innerAppWindowRect innerDocWindowRect isModal	addAnnot addField addIcon bringToFront calculateNow closeDoc createDataObject deletePages embedDocAsDataObject exportAsFDF exportAsFDFStr exportAsText exportAsXFDF exportAsXFDFStr exportDataObject exportXFAData getAnnot getAnnot3D getAnnots getAnnots3D getDataObject getDataObjectContents

Object	Properties	Methods
Document (Continued)	keywords layout media modDate mouseX mouseY noautocomplete nocache numFields numPages numTemplates path outerAppWindowRect outerDocWindowRect pageNum pageWindowRect permStatusReady producer requiresFullSave securityHandler selectedAnnots sounds spellDictionaryOrder subject templates URL viewState xfa XFAForeground zoom zoomType	getField getIcon getLinks getNthFieldName getNthTemplate getOCGs getOCGOrder getPageBox getPageLabel getPageNthWord getPageNthWordQuads getPageNumWords getPageRotation getPageTransition getPrintParams getSound getTemplate getURL gotoNamedDest importAnFDF importAnXFDF importDataObject importIcon importSound importTextData importXFAData mailDoc mailForm openDataObject print removeDataObject removeField resetForm saveAs scroll selectPageNthWord setDataObjectContents setPageAction submitForm syncAnnotScan
Doc.media	canPlay	deleteRendition getAnnot getAnnots getOpenPlayers getRendition newPlayer
Embedded PDF	messageHandler	postMessage

Object	Properties	Methods
Error	fileName lineNumber extMessage message name	toString
event	change changeEx commitKey fieldFull keyDown modifier name rc richChange richChangeEx richValue selEnd selStart shift source target targetName type value willCommit	
Events		add dispatch remove
EventListener		afterBlur afterClose afterDestroy afterDone afterError afterEscape afterEveryEvent afterFocus afterPause afterPlay afterReady afterScript afterSeek afterStatus afterStop onBlur onClose onDestroy onDone onError onEscape onEveryEvent

Object	Properties	Methods
EventListener (Continued)		onFocus onGetRect onPause onPlay onReady onScript onSeek onStatus onStop
Field	alignment borderStyle buttonAlignX buttonAlignY buttonFitBounds buttonPosition buttonScaleHow buttonScaleWhen calcOrderIndex charLimit comb commitOnSelChange currentValueIndices defaultStyle defaultValue doNotScroll doNotSpellCheck delay display doc editable exportValues fileSelect fillColor hidden highlight lineWidth multiline multipleSelection name numItems page password print radiosInUnison readonly rect required richText richValue strokeColor style	browseForFileToSubmit buttonGetCaption buttonGetIcon buttonSetCaption buttonSetIcon checkThisBox clearItems defaultIsChecked deleteItemAt getArray getItemAt getLock insertItemAt isBoxChecked isDefaultChecked setFocus setItems signatureGetModifications signatureGetSeedValue signatureInfo signatureSign signatureValidate

Object	Properties	Methods
Field (Continued)	submitName textColor textFont textSize type userName value valueAsString	
FullScreen	backgroundColor clickAdvances cursor defaultTransition escapeExits isFullScreen loop timeDelay transitions usePageTiming useTimer	
Global		setPersistent subscribe
HostContainer	messageHandler	postMessage
Icon	name	
Icon Stream	read width height	
Identity	corporation email loginName name	
Index	available name path selected	
Marker	frame index name time	
Markers	player	get

Object	Properties	Methods
MediaOffset	frame marker time	
MediaPlayer	annot defaultSize doc events hasFocus id innerRect isOpen isPlaying markers outerRect page settings uiSize visible	close open pause play seek setFocus stop triggerGetRect where
MediaReject	rendition	
MediaSelection	selectContext players rejects rendition	
MediaSettings	autoPlay baseURL bgColor bgOpacity data duration endAt floating layout monitor monitorType page palindrome players rate repeat showUI startAt visible volume windowType	

Object	Properties	Methods
Monitor	colorDepth isPrimary rect workRect	
Monitors	(Same as Array)	bestColor bestFit desktop document filter largest leastOverlap mostOverlap nonDocument primary secondary select tallest widest
OCG	constants initState locked name state	getIntent setAction
PlayerInfo	id mimeTypes name version	canPlay canUseData honors
PlayerInfoList	(Same as Array)	select
PlugIn	certfied loaded name path version	

Object	Properties	Methods
PrintParams	binaryOK colorOverride constants downloadFarEastFonts fileName firstPage flags fontPolicy interactive lastPage nUpAutoRotate nUpNumPagesH nUpNumPagesV nUpPageBorder nUpPageOrder pageHandling pageSubset printAsImage printContent printerName psLevel reversePages usePrinterCRD useT1Conversion	
RDN	c cn o ou e	
Rendition	altText doc fileName type uiName	getPlaySettings select testCriteria
Row	columnArray	
ScreenAnnot	altText alwaysShowFocus display doc events extFocusRect innerDeviceRect noTrigger outerDeviceRect page player rect	hasFocus setFocus

Object	Properties	Methods

`Search`	`attachments`
	`available`
	`bookmarks`
	`docInfo`
	`docText`
	`docXMP`
	`ignoreAccents`
	`ignoreAsianCharacterWidth`
	`indexes`
	`jpegExif`
	`legacySearch`
	`markup`
	`matchCase`
	`matchWholeWord`
	`maxDocs`
	`objectMetadata`
	`proximity`
	`proximityRange`
	`refine`
	`soundex`
	`stem`
	`thesaurus`
	`wordMatching`
		`addIndex`
		`getIndexForPath`
		`query`
		`removeIndex`

`Security`	`handlers`	`getHandler`

`SecurityHandler`	`appearances`
	`digitalIDs`
	`directories`
	`directoryHandlers`
	`isLoggedIn`
	`loginName`
	`loginPath`
	`name`
	`signAuthor`
	`signFDF`
	`signInvisible`
	`signValidate`
	`signVisible`
	`uiName`
		`login`
		`logout`
		`newDirectory`

`SignatureInfo`	(see the *JavaScript for Acrobat API Reference* for a detailed description of the properties)	

Object	Properties	Methods
SOAP	wiredump	connect queryServices resolveService request response streamDecode streamDigest streamEncode streamFromString stringFromStream
Sound	name	play pause stop
Span	alignment fontFamily fontStretch fontStyle fontWeight strikethrough subscript superscript text textColor textSize underline	
Spell	available dictionaryNames dictionaryOrder domainNames languages languageOrder	addWord check checkText checkWord customDictionaryClose customDictionaryOpen ignoreAll removeWord userWords
Template	hidden name	spawn
Thermometer	cancelled duration text value	begin end

Object	Properties	Methods
TTS	available numSpeakers pitch soundCues speaker speechCues speechRate volume	getNthSpeakerName pause qSilence qSound qText reset resume stop talk
util		crackURL iconStreamFromIcon printd printf printx scand spansToXML streamFromString stringFromStream xmlToSpans
XFA	(Corresponds to the appModel container. For details, see *Developing Acrobat Applications using JavaScript*.)	(Corresponds to the appModel container. For details, see *Developing Acrobat Applications using JavaScript*)
XMLData		applyXPath parse

For a complete description of the capabilities and usage of JavaScript for Acrobat, see *Developing Acrobat Applications using JavaScript* and the *JavaScript for Acrobat API Reference*.

3 Interapplication Communication

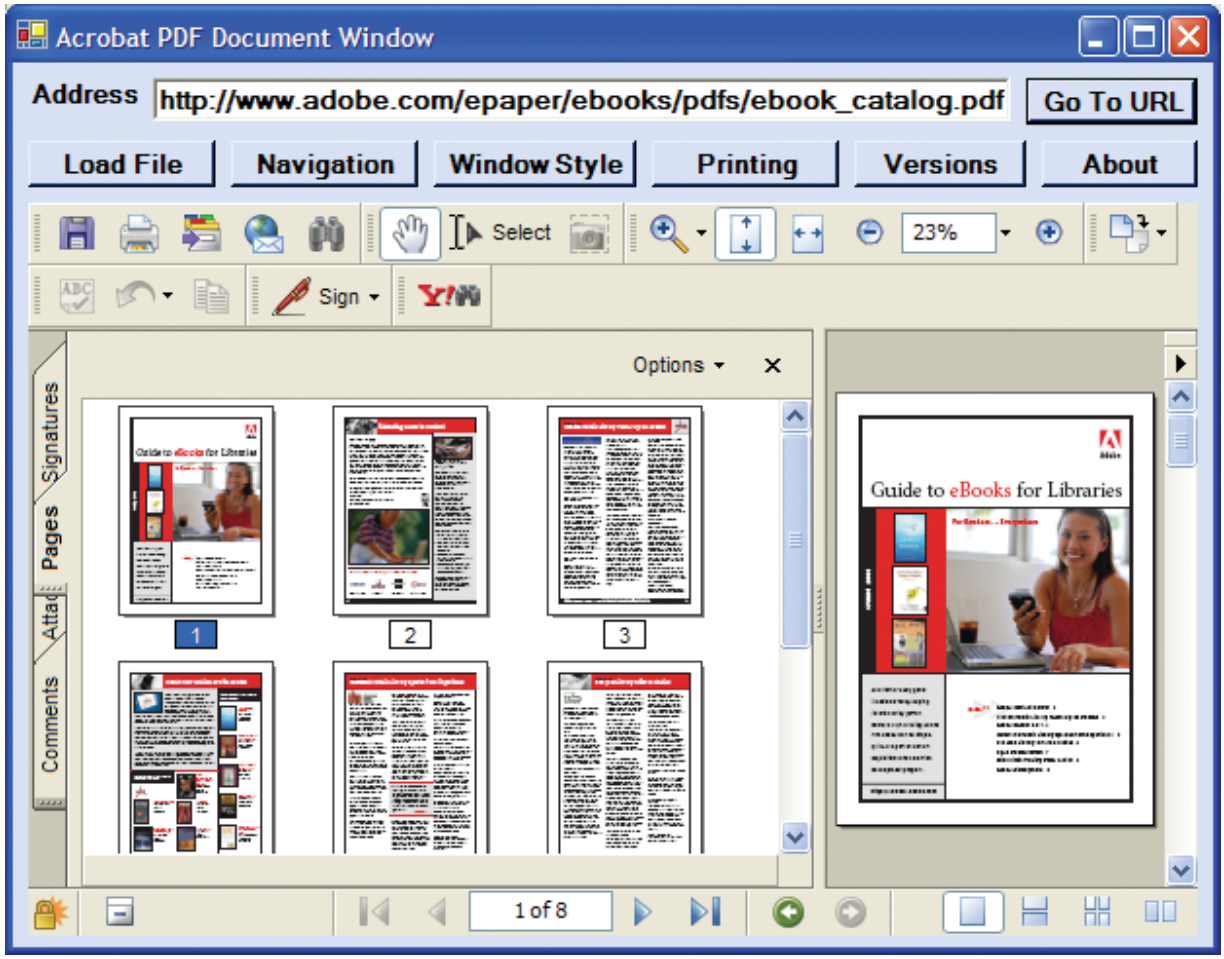
Only a limited subset of the complete IAC functionality is available for Adobe Reader. For detailed descriptions of the syntax and usage of the IAC APIs discussed in this chapter, see the *Interapplication Communication API Reference* and *Developing Applications using Interapplication Communication*.

Note: IAC is not supported on the Linux or UNIX platforms.

OLE automation

On Windows, the only OLE automation supported for Adobe Reader is the *PDF browser controls* interface, which enables you to treat a PDF document as an ActiveX document within an external application. This makes it possible to load a file, move to various pages within the file, highlight a text selection, and specify various print and display options, as shown below.

PDF browser controls



PDF browser controls are available through the `AxAcroPDFLib.AxAcroPDF` interface, which provides the following methods used to programmatically control the PDF document window:

- `GoBackwardStack`
- `GoForwardStack`
- `GotoFirstPage`
- `GotoLastPage`
- `GotoNextPage`
- `GotoPreviousPage`
- `LoadFile`
- `Print`
- `PrintAll`
- `PrintAllFit`
- `PrintPages`
- `PrintPagesFit`
- `PrintWithDialog`
- `SetCurrentHighlight`
- `SetCurrentPage`
- `SetLayoutMode`
- `SetNamedDest`
- `SetPageMode`
- `SetShowScrollbars`
- `SetShowToolbar`
- `SetView`
- `SetViewRect`
- `SetViewScroll`
- `SetZoom`
- `SetZoomScroll`

DDE messages

Adobe Reader supports the following DDE messages:

- `AppExit`
- `CloseAllDocs`
- `DocClose`
- `DocGoTo`
- `DocGoToNameDest`
- `DocOpen`
- `FileOpen`
- `FileOpenEx`
- `FilePrint`
- `FilePrintEx`
- `FilePrintSilent`
- `FilePrintSilentEx`
- `FilePrintTo`
- `FilePrintToEx`

Apple events

On Mac OS, you may use Apple events and AppleScript. Adobe Reader supports only the following *required* Apple events:

- `open`
- `print`
- `quit`
- `run`

4 Plug-ins

The Acrobat core and extended APIs enable you to write plug-ins that integrate with Adobe Reader. For detailed information on the API architecture, methods, and usage, see *Developing Plug-ins and Applications* and the *Acrobat and PDF Library API Reference*.

Any plug-ins written for Adobe Reader must be Reader-enabled, which means that you will need to obtain permission and licensing from Adobe Systems. When developing a Reader-enabled plug-in, follow the steps described in *Developing Plug-ins and Applications* to make specific changes to your plug-in code in order for Adobe Reader to recognize and load it. For information on what you can and cannot do with Reader-enabled plug-ins, see the Reader Integration Key License Program.

A Reader-enabled plug-in is a dynamically linked extension to Adobe Reader created using C/C++ APIs, and can be developed for any supported platform:

- DLLs on Windows (using the extension `.api`)
- Shared libraries (code fragments) on Mac OS X
- Shared libraries on Linux or UNIX platforms

As you learned in "Technologies available within the Acrobat SDK" on page 8, additional usage rights may be applied to a PDF document using LiveCycle Reader Extensions. For information on checking permissions associated with a given PDF document, see *Developing Plug-ins and Applications*.

Note: With LiveCycle Reader Extensions, it is not necessary to distribute any plug-ins or other special software to implement the additional usage rights available within the PDF document.

APIs available for Adobe Reader

Host Function Tables (HFTs) are tables of function pointers, essentially providing a means by which plug-ins call methods in Adobe Reader. The following HFTs are available for development with Adobe Reader:

- `AcroView`
- `AcroViewSweetPea`
- `Cos`
- `PDModel`
- `ASExtra`
- `PDSRead`
- `AcroSupport`
- `Core`
- `Forms`
- `TTS`
- `DigSigHFT`
- `AcroHLS` (Not available on Linux or UNIX platforms)
- `PubSecHFT`
- `Search`
- `WebLink`

For details about specific API support within each of the HFTs, see *Developing Plug-ins and Applications*. For information about the various Acrobat layers and the organization of the related HFTs, see the *Acrobat and PDF Library API Reference*. For a summary of the Acrobat extended APIs, see the *Acrobat and PDF Library API Reference*. For details regarding the additional usage rights that may be applied to a PDF document, see *Developing Plug-ins and Applications*.

Index