Adobe

Adobe Acrobat DC SDK Overview

1/28/21 Adobe, Inc.

Contents

1 Introduction

The Acrobat SDK is a set of tools that help you develop software that interacts with Acrobat technology. The SDK contains header files, type libraries, simple utilities, sample code, and documentation. These tools provide several methods for developing software that integrates with Acrobat products, including JavaScript, plugins, and interapplication communication. These assets should help you design and develop projects such as:

- External applications that communicate with and control Acrobat DC and Acrobat Reader.
- Scripts written in JavaScript that control Acrobat DC and Acrobat Reader.
- Plug-ins that extend Acrobat product functionality.

Developer support

The Acrobat Developer Support team supports software development with the Acrobat DC SDK's core APIs. Developers can purchase support via the Adobe Creative Cloud Exchange Developer Support program. Supported SDK development activities include those for which the product is designed, tested, and licensed. Acrobat Developer Support does not support use cases that do not involve the Acrobat core API.

Only the last two major SDK versions with interim updates are eligible for support.

Note: For non-programmatic issues, such as questions about installing, using, customizing, or deploying Acrobat, contact Acrobat Technical Support

Samples provided with the Acrobat SDK

The Acrobat DC SDK provides a large number of sample applications, plug-ins, and scripts to demonstrate how to use the SDK technologies. Reviewing the samples may guide you in choosing JavaScript, plug-ins, or IAC functionality for a particular implementation. For more information, see the *Acrobat SDK Samples Guide*.

Developing for Acrobat Reader

Acrobat vs. Reader

Acrobat provides a full-featured development environment that includes the entire Acrobat core API. Like Acrobat, the primary technologies for creating software to extend or customize Adobe Reader are JavaScript, interapplication communication, and plug-ins. There are some small differences between the public APIs available in Acrobat DC and Acrobat Reader. Moreover, the APIs that may be used for Acrobat Reader are limited technically and legally.

Both Acrobat and Acrobat Reader accept plug-ins. The primary difference between the two is that, in general, Acrobat Reader can neither make changes to a file nor save a file. API methods that change a file in such a way that a save would be required are not available in Acrobat Reader.

Acrobat Reader plug-in guidelines

If you are considering plug-ins for Acrobat Reader, remember the following:

- You may not develop an Acrobat Reader plug-in without approval from Adobe. There is a web-based application where you describe your plug-in and submit the information to Adobe; Adobe will then review it and let you know whether your application has been approved. The application and the Acrobat Reader Integration Key Licensing Agreement can only be submitted as a web form and are found at http://www.adobe.com/go/acrobat_developer.
- There is a fee to obtain the enabling key.
- There is a restricted set of APIs available in Acrobat Reader.
- For information on how to access the Host Function Tables (HFTs) available to Acrobat Reader plug-ins and how to enable your plugin, see the *Acrobat Plugin Developer Guide.*

Reader enabled plugins

Acrobat Reader only accepts "Reader-enabled" plug-ins, which can access a subset of the APIs available to Acrobat DC. Certain APIs are available to a Reader-enabled plug-in if the PDF document has been assigned additional usage rights (rights-enabled). With a rights-enabled PDF document, the free and ubiquitous Acrobat Reader c an be used to download, save, fill in, digitally sign, and submit electronic PDF forms.

Though the APIs available for Adobe Reader are limited, additional APIs can be used for a given PDF document if that document is rights-enabled, meaning that it has additional usage rights. 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, highlight, and text strikethrough.

SDK technologies and options

You can develop software that integrates with Acrobat DC and Acrobat Reader in three ways: JavaScript, plug-ins, and IAC.

Based on your application's requirements, choose the appropriate technologies for development. In some situations, the desired functionality is only available using one technology. In other situations, you can choose between two or more technologies. For example, you can add menu items to Acrobat DC using either JavaScript or a plug-in. You can also use more than one technology in a single application or single document. For example, you can use both plug-ins and JavaScript to implement a particular feature.

Options include:

- **JavaScript** — Write scripts, either in an individual PDF document or externally, to extend product functionality.

- **Plug-ins** — Create plug-ins that are dynamically linked to and extend product functionality.
- **Interapplication communication** — Write a separate application process that uses interapplication communication (IAC) to control the product. DDE and OLE are supported on Microsoft Windows, and Apple events/AppleScript on Mac OS.
- **PDF manipulation without Acrobat**: You can also use the PDF Library (PDFL) to develop applications that create and manipulate PDF documents but do not interact with Acrobat. For more information, see Adobe PDF Library.

JavaScript

Through its JavaScript extensions, Acrobat DC exposes much of its functionality and its plug-ins to the document author. The JavaScript objects, properties and methods can also be accessed through Visual Basic or C# to automate the processing of PDF documents.

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

JavaScript can be applied at a variety of levels. Each of the levels represents a context in which processing occurs, which affects when the scripts are loaded and how they are accessed inside and outside documents.

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

Plug-ins

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

A plug-in is written in ANSI C/C++ and uses the Acrobat DC public APIs. It can add functionality to Acrobat Pro Extended, Acrobat Professional DC, Acrobat DC Standard, or Acrobat Reader. A plug-in program file goes into a Plug_ins folder or directory and is initialized during Acrobat DC or Acrobat Reader startup.

There are three types of plug-ins:

- **Regular Acrobat DC plug-ins** — These plug-ins run on Acrobat Professional DC and Acrobat DC Standard. Plug-ins for Acrobat Professional DC can use any of the Acrobat DC SDK APIs. Plug-ins for Acrobat DC Standard do not have access to some APIs. For more information, see the *Acrobat Plugin Developer Guide.*
- **Acrobat Reader-enabled plug-ins** — These plug-ins use a restricted set of APIs. Acrobat Reader-enabled plug-ins are developed with permission from Adobe and require special processing to load under Acrobat Reader. Plug-ins for Acrobat Reader can use additional APIs if the PDF document has additional usage rights.
- **Certified plug-ins** — These plug-ins have undergone extensive testing to ensure that they do not compromise the integrity of Acrobat DC’s security model. A checkbox in the Acrobat DC and Acrobat Reader user interface can be used to ensure that only certified plug-ins are loaded. Certified plug-ins can be provided only by Adobe.

Plug-ins are deployed differently on each platform:

- Windows: DLLs. Note, however, that plug-in names must end in .API, not .DLL.
- Mac: Code fragments on Mac OS.

Plug-in development environments

Windows developers can develop plug-ins using C and C++ with Visual Studio.

There is currently no support for development of plug-ins using managed languages such as C# or VB.NET. However, managed languages are supported for use with interapplication communication (IAC). This enables those languages to take full advantage of Acrobat DC's functionality through use of the JavaScript bridge.

All plug-ins developed on Mac OS X must use the Mach-O runtime architecture and must be built as a bundle. Apple Xcode 9.2 is required because SDK projects depend on certain header files that are included with the Xcode development environment.

Acrobat core API

Plug-ins access and control the resources of the Acrobat application host environment using the Acrobat DC core API. The core API consists of a set of methods that operate on objects. The objects have types and encapsulate their data. This object orientation is a conceptual model, implemented using a standard ANSI C programming interface. Methods are C functions; objects are opaque data types. The core API is supported on Windows and Mac.

The API is organized into several layers.

Layer	Description
AV	The AV layer, also known as AcroView or AV Model, works with the Acrobat DC or Acrobat Reader application. Its methods allow plug-ins to manipulate components of the Acrobat DC or Acrobat Reader application itself, such as menus and menu items.
PD	The PD layer, also known as PDModel, provides access to components of PDF documents. Its methods allow plug-ins to manipulate document components such as document pages and annotations.
AS	The AS layer (a support layer) provides platform-independent utility functions and allows plug-ins to override the built-in file-handling mechanisms.
Cos	The Cos Object System layer provides access to the building blocks used to construct documents. Cos methods allow plug-ins to manipulate low-level data such as dictionary and string objects in a PDF file. Whenever possible, you should use higher level APIs to access and manipulate PDF files. Though you can use the Cos layer APIs to perform most types of access or manipulation of a PDF file, it can be difficult and requires in-depth knowledge of PDF file structure.

The core API also includes platform-specific plug-in utilities to handle issues that are unique to Windows and Mac. For more information, see the *Acrobat Plugin Developer Guide.*

Extended APIs for plug-ins

Plug-ins can expose their own functionality and make it available to other plug-ins in the same way that Acrobat DC functionality is available through the core API. Acrobat DC uses many plug-ins to implement features, such as the Search and Digital Signature plug-ins. In fact, the Acrobat DC architecture encourages the use of plug-ins to expose APIs for use by other plug-ins.

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

- Catalog
- Digital Signature
- Forms
- PDF Consultant
- Search
- Spelling
- Weblink
- SaveAsXML

For more information on plug-ins and HFTs, see the the *Acrobat Plugin Developer Guide* and the *Acrobat and PDF Library API Reference*.

JavaScript vs. plugins: pros and cons

While developers writing plug-ins have direct access to the Acrobat Core API, JavaScript applications tend to be easier to write and implement, since they are developed using the editor and debugger that are included in Acrobat. JavaScript applications are also easier to distribute since they can be included directly within a PDF file, whereas plug-ins must be placed in the Plug_ins folder by either an installer or the user. JavaScript for Acrobat can be used across multiple platforms, while a plug-in must have separate versions for each platform in order to handle certain platform-specific issues.

In general, plug-ins allow for more direct control over Acrobat DC than JavaScript. There is a richer set of APIs that you can use from a plug-in. Since it is interpreted rather than compiled, execution of JavaScript for Acrobat code is slower than plug-in code. However, the difference tends to be noticeable only in computationally intensive applications, such as a full text search in a large PDF file.

Implementation comparison

Comparison of plug-ins and JavaScript

	Plug-ins	JavaScript
Scope	A plug-in affects all PDF documents viewed by Acrobat DC.	JavaScript can affect either a single document or all PDF documents.
Installation and distribution	Plug-ins must be placed in the Plug_ins folder or directory by an installer or by the user.	Document-level JavaScript code is easier to distribute since it can be included directly within the PDF file and does not require an installer. Folder-level JavaScript code must be placed in the Acrobat DC application JavaScript folder or the user's JavaScript folder.
Low-level access	Plug-ins can access and manipulate low-level objects in the PDF object model, such as the Cos layer.	JavaScript can access a limited set of AV and PD layer objects.
Execution speed	Plug-ins are compiled and loaded when Acrobat DC initializes.	Execution of JavaScript code is slower than plug-in code because it is interpreted instead of compiled. However, the difference is noticeable only in computation-intensive applications, such as a full-text search in a large PDF file.
Ease of implementation	Plug-ins are developed in C or C++ and are compiled and linked in the appropriate development environment. You must include all necessary header files for your application.	JavaScript scripts are easier to write and implement since they are developed using the editor and debugger that come as part of Acrobat Professional DC. Developers can also use an external editor to create and edit JavaScript code.
Cross-platform compatibility	Plug-ins must be built on different platforms to handle certain platform-specific issues.	JavaScript is cross-platform compatible.

Feature comparison

Finally, while some JavaScript and plug-in capabilities overlap, others are only available in JavaScript while others are only available with a plug-in as summarized in the table below. JavaScript for Acrobat is well-suited to quickly tasks such as adding user interface capabilities, forms processing, interacting with databases and web services, and so on.

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

Capability	JavaScript	Plug-in
Use SOAP and web services	Yes	No
Manipulating multimedia	Yes	No

Capability	**JavaScript**	**Plug-in**
Automate email review workflow	Yes	No
Search Acrobat Help	Yes	No
Use Acrobat security policies	Yes	No
Use content stream and other low-layer access	No	Yes
Add content to a PDF file by using a content stream	No	Yes
Create new menus or toolbars	No	Yes
Create new annotation or action types	No	Yes
Modify the ASFixed scaling factor for large PDF file sizes	No	Yes
Access platform-specific services or events	No	Yes
Getting and setting wireframe drawing mode	No	Yes
Accessing Cos and other low-layer objects	No	Yes

Interapplication communication

To take advantage of Acrobat DC functionality from within an external application, use interapplication communication (IAC). Acrobat DC provides support for IAC through OLE automation and DDE on Windows as well as Apple events and AppleScript on Mac OS. Acrobat Reader also supports IAC, but does not support OLE on Windows.

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

On Windows, you can develop IAC applications using Visual Basic .NET, Visual C++ .NET, or Visual C# .NET. On Mac OS, you develop IAC applications using Xcode. CodeWarrior is not supported.

For more documentation, see the *IAC Developer Guide*.

Viewing PDF documents from an external application

If your Windows application only views a PDF document and does not need to edit it in any way, use the PDF Browser Controls to view the document from your external VB or C# application. When you open a document for viewing using the PDF Browser Controls, the document is displayed in the application window. Acrobat DC toolbars are also displayed and can be used with no additional API calls. The toolbars can be hidden. Acrobat DC or Acrobat Reader must be installed for the PDF Browser Controls to work.

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

Controlling Acrobat from an external application

If you need to do more than just view a PDF document from your application, you can use the IAC API to perform these tasks:

- Get annotations, text and form data from a PDF document
- Search a PDF document
- Manipulate a PDF document, editing and adding content

Control Acrobat DC (but not Acrobat Reader) remotely

Plug-ins for IAC

You can extend the functionality of the IAC interfaces by writing plug-ins that use core API objects that are not already part of the IAC support system. The Acrobat DC SDK provides a sample that demonstrates this. For more information, see the *Acrobat SDK Samples Guide*.

JavaScript support

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

Windows support

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

On Windows, you can display a PDF document in applications using simplified browser controls. In this case, the PDF is treated as an ActiveX document, and the interface is available in Acrobat Reader.

Once the PDF document is loaded, you can implement browser controls to perform the following tasks:

- Determine which page to display
- Control view and zoom modes
- Determine whether to display bookmarks, thumbnails, scrollbars, and toolbars
- Print pages using various options
- Highlight a text selection

Apple event support

The Acrobat DC viewers support Apple events and a number of Apple event objects on Mac OS. IAC support includes some of the objects and events described in *Apple Event Registry: Standard Suites*, as well as Acrobat DC-specific objects and events.

You can find information on Apple events supported by the Acrobat DC Search plug-in by referring to the *Acrobat and PDF Library API Reference*. Other plug-ins supporting additional Apple events are described in *Acrobat Plugin Developer Guide*.

Adobe PDF Library

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

Designed specifically for OEMs, ISVs, system integrators, and enterprise IT developers, the Adobe PDF Library SDK contains a set of functions for developing third-party solutions and workflows around PDF. The library enables PDF functionality to be seamlessly embedded within applications without the presence of Acrobat DC or Acrobat Reader. It also provides a reliable, accurate, and Adobe-supported implementation of the latest PDF specification.

There is significant overlap between the functionality provided by the PDF Library SDK and the Acrobat DC SDK. They differ in providing access to the Acrobat DC user interface:

- The Acrobat DC SDK is designed for Acrobat product environments and allows you to control and interact with the Acrobat DC user interface.
- The PDF Library SDK is intended for interaction between PDF and other applications, such as high volume batch processing and PDF generation applications. It does not export methods for creating or managing Acrobat DC user interface elements—that is, the AcroView (AV) layer of the core API.

For more documentation, see the *IAC Developer Guide*.

1 Sandbox Broker Extensibility

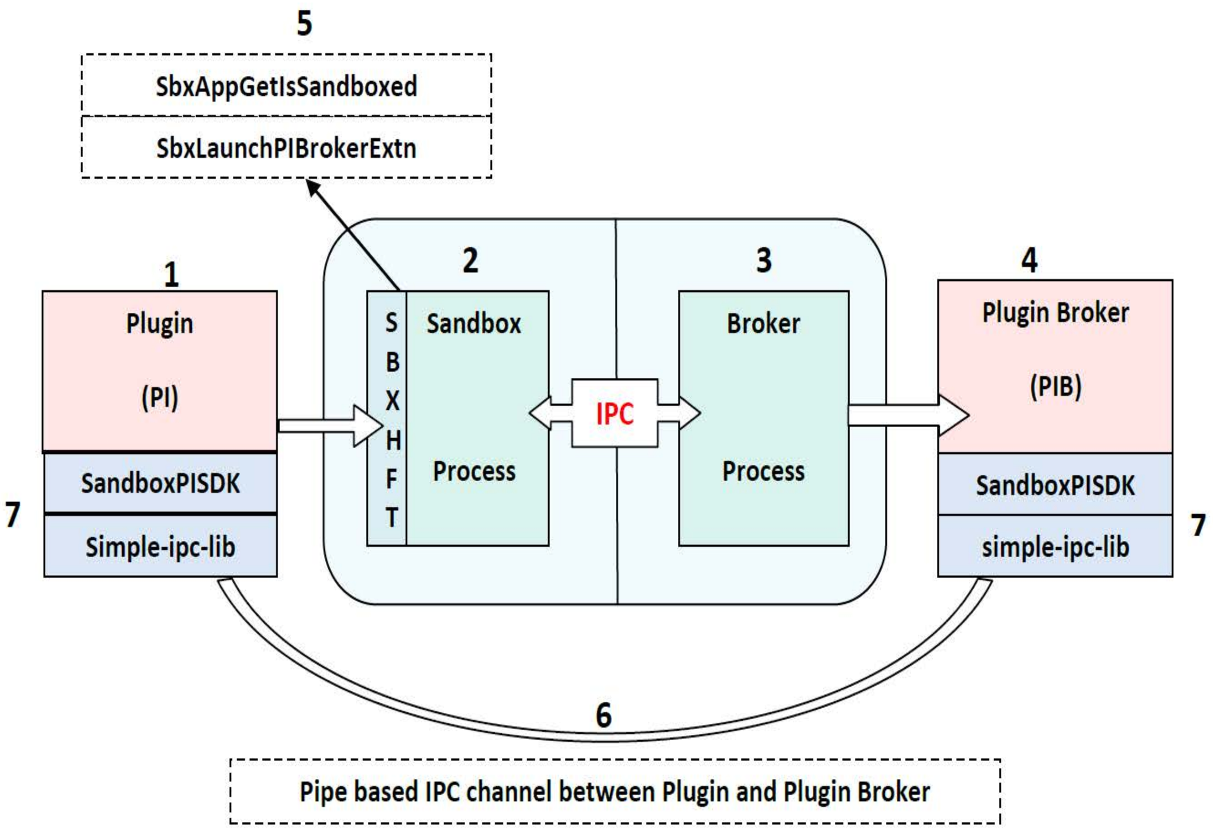
A plug-in loaded in a sandboxed process executes within the restrictions of the sandbox and it may need to implement a broker to perform any privileged operations or access resources that are blocked by the sandbox. This might require adding to the existing broker APIs to get its features working in the sandbox environment.

Broker extensibility provides a generic plug-and-play architecture that would enable extending the existing broker functionality provided by Protected Mode in the app. This architecture allows new features to work seamlessly in the sandbox environment on demand and at run time.

Extending broker APIs at run time

The `sandbox_pisdk_client` and `sandbox_pisdk_server` libraries have been made available for broker extensibility. Plugins need to link to this library to implement broker extensibility. Doing so allows plug-ins loaded in sandbox process to extend broker APIs at run time and make IPC calls to the broker process to access the extended functionality.

The following figure illustrates the sandbox plug-in architecture provided to extend the broker API functionality by individual plug-ins at run time:



Components

Plugin (PI)

The plug-in is loaded in the sandbox process that runs under restricted rights. Every plug-in that requires some added broker functionality must implement its own plug-in broker process to implement new broker APIs. These broker APIs can be used only by that plug-in and are not visible to any other code/module loaded inside the sandbox process. Plugins can use the sandbox plugin SDK for this.

Sandbox process

This is the renderer process running in protected mode under restricted rights. All application and plug-in code runs in this process. It has restricted access to files, registry, processes, window handles etc. It communicates with the broker process over the sandbox-broker IPC channel to service actions outside the sandbox boundary.

Broker process (The broker process running with full rights)

This is the broker process which runs with full user rights and imposes policy white lists. It grants access to files, registry, and processes as per policy white list. It implements the broker APIs to service the functionality requested by the sandbox process outside the sandbox boundary. The sandbox process communicates with it over the sandbox-broker IPC channel.

Plugin broker (The plugin broker process that extends the broker APIs at runtime)

This is the plugin broker process which is private to the associated plug-in. This implements the plug-in specific broker APIs required for the full functionality of the plug-in in sandbox environment. We call them extended broker APIs, as they extend the existing broker APIs to fulfill the plug-in requirements.

The plug-in, which runs in Protected Mode context, communicates with this process directly over a separate IPC channel, 1 -> 6 -> 4, to service additional functionality outside the sandbox boundary which is not provided by broker APIs.

This process is launched by the broker process; it runs with full user rights and independent of the sandbox context. Also, by default it does not honor any policy restrictions or other security mechanisms. Thus it is the plug-in developer's responsibility to design and implement it with security in focus; otherwise it can be used to escape the sandbox context.

Also, this binary along with the description (input and output parameters) of the APIs it exposes have to be submitted to Adobe for code signing. Please note that signature of the plug-in broker executable is verified before launch if the 'Use only Certified plug-ins' is checked, so it is the responsibility of the plug-in developer to ensure that this is signed otherwise this would just not work.

SandboxHFT (Public HFT provided by sandbox for broker extensibility)

This HFT contains two APIs which can be called by plug-ins to implement broker extensibility:

- `SbxAppGetIsSandboxed`: This public API can be called from any plug-in that imports SandboxHFT to check whether the app is running with Protected Mode ON or OFF.
  The prototype for this API is:
  `ASBool SbxAppGetIsSandboxed(void);`

- `SbxLaunchPIBrokerExe`: This is the public API provided by sandbox process to extend broker API at run time. The plug-in makes a request via the `SbxLaunchPIBrokerExe` API to start its plugin broker process and do some initial handshake with it.

All the processes placed in `App Install Directory\(app name)\plug_ins\pi_brokers` are white listed. Third party plug-in developers must place their plugin broker processes at this location and can use this HFT. The prototype for this API is:

```
HANDLE SbxLaunchPIBrokerExe(
const wchar_t *piBroker // IN:plugin broker process name
);
```

It takes the plugin broker process name in `piBroker` as input, looks it up in its process white list, if it succeeds then launches the piBroker process, and finally establishes IPC channel between the plug-in and piBroker. If everything succeeds, it then returns handle to the underlying IPC mechanism (in this it's the `NamedPipe`) otherwise INVALID_HANDLE_VALUE is returned.

To aid thrid-party developers in setting up the IPC between plugin and plugin broker process (1 -> 6 -> 4), we have provided an API `SandboxPISDK::InitSandboxPIBrokerIPCChannel` which launches the PI-broker (by using above HFT function `SbxLaunchPIBrokerExe`) and sets up the IPC. See Building a plug-in for more details.

Please note that there's one more API in this HFT - SbxLaunchPIBrokerExtn which has been deprecated from A11

IPC channel between plugin and plugin broker, 1->6->4

This is the dedicated IPC channel across which the plug-in and its plug-in broker process communicate. The IPC channel is established at run time, when the plugin makes a request via the `InitSandboxPIBrokerIPCChannel` API, to start its plugin broker process. The broker process looks up in its process white list and launches the requested plugin broker process and establishes the IPC channel. This communication channel bypasses the usual 1 -> 2 -> 3 trusted IPC channel between the plugin (in Protected Mode process) and broker process and talks directly to the PI broker process via 1->6->4.

simple-ipc-lib and SandboxPISDK

`simple-ipc-lib`: This is a very light weight C++ inter-process communication library. It is used to communicate over the `NamedPipe`. This library needs to be built and linked into both the plugin and the plugin broker.

`SandboxPISDK`: plugin developers must use this SDK to implement broker Extensibility for their Plugin. This SDK abstracts out a lot of internal details and has wrappers and macros to aid developers.

Building a plug-in

1. Add ipc_lib project to the plug-in's project dependency list and also link to it.
2. Add the following headers from the SandboxPISDK into your project:
   - `SandboxPIIPCDefs.h`
   - `SandboxPIClientDefs.h`
   - `SandboxPISDKClientMacros.h`

3. Implement `<plugin>CrossCalls.h` and `<plugin>IPCClient.cpp` to add the client side of the Plugin specific broker Extensibility functionality. See Implement CrossCalls (make calls from plug-in to plug-in's broker process).

4. Link to the `sandbox_pisdk_client.lib` provided along with SandboxPISDK

5. Call the API, `InitSandboxPIBrokerIPCChannel` provided with SandboxPISDK to setup the IPC channel with your Plugin's broker process.

Building a plugin's broker process

1. Add `ipc_lib` project to the plug-in's project dependency list and also link to it.

2. Add the following headers from the SandboxPISDK into your project:
   - `SandboxPIIPCDefs.h`
   - `SandboxPIServerDefs.h`
   - `SandboxPISDKServerMacros.h`

3. Add the `SandboxPIServerMain.cpp` to the project

4. Implement `<plugin>IPCServer.cpp` to add the server/broker side of the Plugin broker Extensibility functionality. See Implement CrossCalls (make calls from plug-in to plug-in's broker process).

5. Add `ipc_lib` project to your <Plugin>broker project's dependency list and also link to it.

6. Link to the `sandbox_pisdk_server.lib` provided along with SandboxPISDK

Refer the sample plugin/pluginbroker source code for more details.

Handshake between plugin and plugin broker

These steps create the initial setup for the IPC channel between the Plugin and its broker required to make the cross calls. The following sequence of steps takes place during the initial setup:

1. The plug-in provides a plug-in broker process placed at `(app name)\plug_ins\pi_brokers`.

2. The plug-in sends a request via the `InitSandboxPIBrokerIPCChannel` call which in turn uses SandboxHFT call `SbxLaunchPIBrokerExe` to launch its piBroker process.

3. The broker process then validates the `piBroker` process name against a white list.

4. If the above validation succeeds, then the broker process launches the `piBroker` process. It then establishes the IPC channel using the `simple-ipc-lib` between `piBroker` process and plug-in.

5. If `InitSandboxPIBrokerIPCChannel` returns true then plug-in is now ready to make cross calls to the `piBroker` directly over the IPC channel to execute extended broker functionality.

Refer to the sample project to get started and for example code.

Implement CrossCalls (make calls from plug-in to plug-in's broker process)

The entire purpose of providing broker Extensibility to plugins is to offer plug-ins a mechanism to make CrossCalls i.e. perform privileged operations that don't work when called directly from the plug-in, for example, sending messages to HWND, read/write to files/registries, launch any application etc. The plug-in's broker run under the user rights and doesn't have any sandbox restrictions as a plugin running inside the app process has. So when a CrossCall is made, the plugin's broker performs the same operation on the plugin's behalf and returns the result to plugin. We are using chrome's `simple-ipc-lib` to perform Inter-process Communication and have written macros to define and make CrossCalls relatively easier.

Let's take an example where plugin needs to send `WM_COPYDATA` message to a window.

Changes required in the plugin (client)

File <plugin>CrossCalls.h

Create a unique id for new call and add it in between `DECLARE_CROSS_CALL_START` and `DECLARE_CROSS_CALL_END`:

```
DECLARE_CROSS_CALL_START

          CROSS_CALL_ID(<UniqueId1>)

          CROSS_CALL_ID(<UniqueId2>)

          CROSS_CALL_ID(<UniqueIdN>)

          CROSS_CALL_ID(SbxPISendMessage)

DECLARE_CROSS_CALL_END
```

Now the `INPUT PARAMETERS` (params that are passed from plugin to broker) and `OUTPUT PARAMERERS` (params that are returned from broker to plugin) and their order need to be defined.

Macros `DEFINE_IPC_MSG_CLIENT(<UniqueId>, <numOfInputParams>)` and `DEFINE_IPC_MSG_SERVER(<UniqueId>, <numOfOutputParams>)` have been designed exactly for this purpose. For `SbxPISendMessage` CrossCall, the 4 input parameters are `HWND` (passed as `VoidPtr`), `MsgType` (passed as `UINT`), `WPARAM` (passed as `UINT`), and `buffer` (passed as `ByteArray`), and 2 output parameters are result – retrun value of `SendMessage` call (passed as `UINT`) and `LastError` – `GetLastError` result after `SendMessage` (passed as `UINT`).

```
DEFINE_IPC_MSG_CLIENT(SbxPISendMessage, 4) {

          INPUT_PARAM_VOID_PTR(1)

          INPUT_PARAM_UINT(2)

          INPUT_PARAM_UINT(3)

          INPUT_PARAM_BUFFER(4)

};
```

```
            DEFINE_IPC_MSG_SERVER(SbxPISendMessage, 2) {

                      OUTPUT_PARAM_UINT(1)

                      OUTPUT_PARAM_UINT(2)

};
```

The following IPC data types are supported (as defined in files `SandboxPISDKClientMacros.h` and `SandboxPISDKServerMacros.h`) :-

```
INPUT_PARAM_VOID_PTR - for void*
INPUT_PARAM_INT - int
INPUT_PARAM_UINT - unsigned int
INPUT_PARAM_LONG - long
INPUT_PARAM_ULONG - unsigned long
INPUT_PARAM_STRING - std::string
INPUT_PARAM_WSTRING - std::wstring
INPUT_PARAM_BUFFER - ipc::ByteArray
OUTPUT_PARAM_VOID_PTR - for void*
OUTPUT_PARAM_INT - int
OUTPUT_PARAM_UINT - unsigned int
OUTPUT_PARAM_LONG - long
OUTPUT_PARAM_ULONG - unsigned long
OUTPUT_PARAM_STRING - std::string
OUTPUT_PARAM_WSTRING - std::wstring
OUTPUT_PARAM_BUFFER - ipc::ByteArray
```

File <plugin>IPCClient.cpp

In this file, the entire client side of the CrossCall needs to be defined. Macro `DEFINE_CLIENT_CROSS_CALL(<UniqueId>, <numOfInputParams>, <numOfOutputParams>)` does all the work of generating the requisite class (client side).

```
DEFINE_CROSS_CALL_CLIENT(SbxPISendMessage, 4, 2)
```

So once you have defined the cross call by using above macro, you have to use macro `MAKE_CROSS_CALL() and MAKE_CROSS_CALL_WITH_RETURN_VAL()` to call the function at the broker end. Whenever any UI is involved at the broker end and window handle from plugin is passed to the `PIBroker`, the macro `MAKE_CROSS_CALL_WITH_MSG_PUMP()` or `MAKE_CROSS_CALL_WITH_MSG_PUMP_AND_RETURN_VAL()` can be used which ensures that all UI messages from `PIBroker` to the plugin. Remember, the order of input and output parameters have to be exactly same as defined in `<plugin>CrossCalls.h` header file. Also, note that the input parameters are passed as const reference and output parameters are passed as reference.

```
           bool return_value = false;

           MAKE_CROSS_CALL_WITH_RETURN_VAL(SbxPISendMessage,
return_value)(static_cast<LPVOID>(hWnd), msgType, wParam, ByteArrayBuffer,
outRetVal, outLastErr);
```

Changes required in the plugin's broker (server)

File <plugin>IPCServer.cpp

In this file, the entire server side of the CrossCall needs to be defined. Macro `DEFINE_SERVER_CROSS_CALL_START(<UniqueId>, <numOfInputParams>, <numOfOutputParams>)` does all the work of generating the requisite server side class:

```
            DEFINE_SERVER_CROSS_CALL_START(SbxPISendMessage, 4, 2)

                virtual void Proc(const LVOID& input1, const UINT& input2, const
UINT& input3, ipc::ByteArray &input4, UINT& out1, UINT& out2)

                {

                   LRESULT result = 0;

                   HWND hWnd = static_cast<HWND>(input1);

                   //  likewise take other parameters and make them ready to be
passed to

                   // SendMessage

                   result = SendMessage(hWnd, WM_CPYDATA, <other parameters>);

                   out1 = (result == FALSE) ? 0 : 1;

                   out2 = GetLastError();

                 }

           DEFINE_CROSS_CALL_SERVER_END
```

Please note that if Proc above raises any exception then it will be eaten away by the internal implementation, so you need to use your own try catch block if you want to pass on any exception raised here to the plug-in.

And finally use the `MACRO - ADD_MSG_HANDLER` to add above Cross all Proc to list of message handlers so that whenever `SbxPISendMessage` message arrives at the `PIBroker` end, it can be appropriately routed to its Proc.

```
        DEFINE_CROSS_CALL_HANDLER_START(<Plugin-Name>)
              ADD_MSG_HANDLER(<Cross-Call UniqueId1>)
              ADD_MSG_HANDLER(<Cross-Call UniqueId2>)
              ADD_MSG_HANDLER(SbxPISendMessage)
         DEFINE_CROSS_CALL_HANDLER_END
```

Individual policy configuration for 3rd parties

The 2020 release introduces a new mechanism which allows plugins to use individual custom policy config files. The new config files adhere to same format and syntax as the original config file. To create one:

1. Create a custom policy config file parallel to Acrobat.exe in Acrobat's installation folder.
2. Open the registry to `HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Adobe\Acrobat Acrobat\<Track>\FeatureLockDown\`
3. Create a key: `cProtectedModeConfigFiles`
4. Add new preferences with a string with the name as your plugin's name and data as your config file name under:

Name	**Type**	**Value**
`t<plugin-name>.api`	REG_SZ	<config-filename>.txt

Getting started with SampleExtn

To use the example, do the following:

1. Copy the `SampleExtn` folder to the ...`\[PluginSupport]\Sample` folder of your `SDK` folder as: `<Acrobat_SDK_Root>\Adobe\AcrobatSDK\Version1\[PluginSupport]\Samples\[SampleExtn]`. This folder contains the source code for SamplePI(Reader Plugin) and SamplePIBroker(PIBroker).
2. Open the Solution File: `SampleExtn\win32\[SampleExtn].sln` and build it with VisualStudio 2019 SP1. Note: You will have to Reader Enable the `SamplePI.api` as described in Developing and enabling an Adobe Reader plug-in--Reader plugins must be signed; Acrobat-only plugins do not need signing.
3. Copy the Reader Enabled `SamplePI.api` to `<Reader_Installed_Path>\plug_ins\[SamplePI].api`
4. Copy the SamplePIBroker.exe process to `<Reader_Installed_Path>\plug_ins\pi_brokers\[SamplePIBroker].exe`
5. Start in Protected Mode and play with `SamplePI.api` from Acrobat SDK/SamplePI Menu.

1 PDF File Creation

This chapter provides information about how you can use the Acrobat DC SDK to create PDF files.

Creating PDF files from an authoring application

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

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

Acrobat Distiller

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

On Windows, you can automate Distiller through OLE automation (commonly referred to as ActiveX or COM). The automation interface makes it easy to start and control Distiller from any programming language that supports automation. Distiller supports programming environments written for Visual Basic and for Visual C++ with and without MFC.

The Mac OS version of Distiller supports Apple events. Apple events can be used from external applications written in programming languages such as C or from AppleScript. Because AppleScript is more straightforward, it is recommended.

For more information on Distiller, see the Distiller Help and the Acrobat Distiller API Reference

Automated creation of PDF documents from Windows

The Acrobat DC SDK has a set of APIs that allow you to automate the creation of PDF files from an external Windows application.

When you use OLE automation, the Distiller splash screen and status are displayed even when the process is automated. With the API, there is no interaction with the user and no display on the screen. The API allows you to specify the name and location of the output PDF file. To do so, you must modify the Windows registry. For more information, see the Acrobat Distiller API Reference.

Automatic generation of advanced Acrobat features

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

For more information, see the *pdfMark Reference*.

Attaching a native document to a PDF file

Another way that an authoring application can integrate with Acrobat DC or Acrobat DC Reader is to allow the user access to the original authoring document through the Acrobat DC user interface. Through the use of private data in a PDF file, an authoring application can embed the entire authoring document as part of the PDF file that represents it. This way, not only can the resulting electronic document be viewed by anyone using Acrobat DC, it can also be edited by users who have the authoring application.

You must write a plug-in for Acrobat DC to allow users to embed and extract the authoring document. This plug-in would add the authoring document as a private data stream when embedding, and, when extracting, save the stream to a temporary file and invoke the authoring application.

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

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

Batch processing with Distiller

You do not need to use the Distiller API to integrate Distiller with your product. Distiller has the ability to watch directories over a network and to convert any PostScript files saved to those directories to PDF. It is also possible to set different job options for each directory so that one directory can be used, for example, for high-end color output, while the other can generate a more compressed file suitable for web use. These features of Distiller are not supported by Acrobat Developer Support. Check the help documentation packaged with the product or books by Adobe Press and other publishers for using Distiller through the user interface.

The Distiller API can be used to programmatically process files and set the output path and file names while Acrobat Professional DC can be used to set up watched folders.

Tagged PDF documents

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

Tagged PDF is a particular use of structured PDF that allows page content to be extracted and used for various purposes, including:

- Reflow of text and graphics
- Conversion to file formats such as HTML and XML
- Access for the visually impaired

Creating PDF files using plug-ins or JavaScript

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

Empty PDF files

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

JavaScript provides support for dynamic PDF file creation and content generation. Using a plug-in, you can also create a new, empty PDF file. You can then use methods to add content to the created file.

PDF files from multiple files

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

Using JavaScript, you can import an external file into a PDF document. After doing this, it is possible to extract information from the data object for placement and presentation within the PDF document. You can automate the insertion of multiple PDF files into a single PDF document through a series of method calls, and you can also use a portion of the current document to create a new document.

With a plug-in or IAC, you can extract data from other file types, and then bind the resulting PDF files into one PDF file. The relevant APIs can also be executed directly from an IAC application.

Creating PDF files without using Acrobat

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

The best resource for building PDF-generation capability is the *PDF Reference*, which fully documents the PDF specification. The PDF file format is complex, and developing code to generate it requires a significant amount of development.

1 Working with PDF Features

This chapter discusses how you can work with various features of PDF documents using the Acrobat DC SDK.

Navigation in PDF documents

Bookmarks

A bookmark corresponds to an outline object in a PDF document. A document outline consists of a tree-structured hierarchy of bookmarks

that displays the document's structure, allowing the user to navigate interactively from one part of the document to another. Each bookmark has a title that appears on screen and an action that specifies what happens when the user clicks on the bookmark.

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

You can use JavaScript, a plug-in, or IAC to get or set the following properties:

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

Thumbnails

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

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

You can use the Acrobat DC SDK to create and remove thumbnails in a PDF document.

Links

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

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

In addition, JavaScript can be used to specify whether cross-document links are opened in the same window or in a new one.

Actions for special effects

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

You can also customize these actions so that they change their appearance after the user has clicked on them.

PDF page manipulation

You can use the Acrobat DC SDK to insert or remove pages from a PDF document. For example, you can do the following tasks:

- Create an empty page in the current document (not with IAC)
- Insert pages from another document into the current document
- Move a page to another location in the same document (not with IAC and AppleScript)
- Replace pages with pages from another document
- Delete pages from the current document

You can also access JavaScript functionality from an external application.

Page content

Page content is a major component of a PDF file. It represents the visible marks on a page that are drawn by a set of PDF marking operators. The set of marking operators for a page is also referred to as a *display list*, since it is a list of marking operations that represent the displayed portion of a page. For more information on page content streams, see the *PDF Reference*.

You can use the PDFEdit API to access PDF page contents. With PDFEdit, your plug-in can treat a page's contents as a list of objects rather than manipulating the content stream's marking operators. There is no JavaScript equivalent to the PDFEdit API to allow you to manipulate page content. For more information on this API, see the *Acrobat and PDF Library API Reference* and the *Snippet Runner Cookbook*.

Document logical structure

You can insert logical structure into a PDF document by creating a tagged PDF document. The PDSEdit API provides the ability to add, modify, and view this logical structure. For more information, see the *Acrobat and PDF Library API Reference*.

Authoring applications can also add structure pdfmarks to the PostScript language code generated when a document is printed. For more information, see the *pdfmark Reference*.

Other ways of modifying PDF documents

You can use a plug-in or JavaScript to modify a PDF document by cropping and rotating pages, numbering pages, and adding headers and footers.

Watermarks

JavaScript provides methods to create watermarks within a document, and place them in optional content groups (OCGs). You can also add watermarks using C APIs. The Acrobat DC SDK contains a sample plug-in that adds a watermark. For more information, see the *Acrobat SDK Samples Guide.*

Spell-checking

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

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

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

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

Multimedia

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

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

For more information, see *Acrobat JavaScript Developer Guide.*

Printing PDF files

Using the Acrobat DC SDK, you can control the way that Acrobat DC, Acrobat Reader, or your external application prints PDF files. Using any of the Acrobat DC technologies, you can customize the way that a PDF document is printed.

Since printing involves sending pages to an output device, there are many options that can affect print quality. JavaScript can be used to enhance and automate the use of these options in print production workflows. For more information, see the *Acrobat JavaScript Developer Guide*.

With plug-ins, you can use methods to customize and control how a PDF file is printed from Acrobat DC or Acrobat Reader. Depending on the methods you use, a user interface may be invoked. For example, you can use a method to print a document to the current printer using the current print settings and job settings with no user interface. You can specify a page range, a PostScript version, and whether to shrink the pages to fit the printer. For more information, see *Acrobat Plugin Developer Guide*.

Using the IAC APIs, you can print a PDF file from an external application. For more information, see *Interapplication Communication Developer Guide*.

Embedded fonts

Acrobat Distiller and the PDF Library add font embedding information to fonts that are embedded in PDF files. With the inclusion of this information, your code can determine how an embedded font can be used. These operations also may apply to code used with the PDF Library SDK.

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

PDF Library users can perform these operations using an existing PDF document, or they can create a PDF document from scratch that includes embedded fonts. Creating a document from scratch cannot be performed by a plug-in, but it can be done by using PDF Library calls from within a compiled application that includes the PDF Library.

1 User Interface Modifications

This chapter describes how you can modify menus and toolbars in the Acrobat DC or Acrobat Reader user interface using the Acrobat DC SDK.

Menu items and menus

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

Menu items

You can use a plug-in or JavaScript to add or remove menu items or submenus to or from an existing menu. With IAC OLE automation, you can remove a menu item but you cannot add one.

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

You are encouraged to have your plug-in add its menu items to the Tools menu. When it is launched, Acrobat DC or Acrobat Reader automatically adds this menu, as well as the About Plug-ins and Plug-in Help menus (see "Plug-in help files" on page 8). After Acrobat DC or Acrobat Reader loads all plug-ins, it checks these three menus and removes any that are empty.

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

Menus

You can create a new menu from a plug-in. However, you cannot create menus or add menus to menu bars using JavaScript or IAC.

Toolbars

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

Items on a toolbar

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

Toolbar creation

Plug-ins also can create new toolbars, called flyouts, that can be attached to buttons on the main toolbar. The selection tools, for example, are all on a flyout. Not all tool buttons are located on the main toolbar; some may be located on a flyout.

You cannot create a flyout with JavaScript or IAC.

Customization of Acrobat Help

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

About dialog box and splash screen

Plug-ins can set values in the preferences file to prevent the Acrobat DC or Acrobat Reader About dialog box or splash screen from appearing before displaying the first document. These changes take effect the next time Acrobat DC or Acrobat Reader is launched.

Plug-in help files

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

You can place a help file either in the Help directory or in a subdirectory of the Help directory. If, for example, your plug-in is localized for Japanese, you can place the Japanese help file in its own subdirectory. For more information, see the *Acrobat and PDF Library API Reference.*

1 Annotations and Online Collaboration

Acrobat DC 8.0 introduces the Shared Reviews feature, which you should use for most online collaboration processes. You cannot initiate a Shared Review using the Acrobat DC SDK. For information on Shared Reviews, see the Acrobat DC Help.

About annotations

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

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

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

Annotations and JavaScript

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

Using JavaScript, you can perform the following tasks:

- Add note comments
- Make text edits
- Highlight, cross out, or underline text
- Add or delete custom stamps
- Add comments in a text box
- Add attachments
- Spell check in comments and forms
- Add commenting preferences
- Change colors, icons, and other comment properties

Annotations with plug-ins or IAC

There is an abstract superclass for all annotations. Acrobat DC and Acrobat Reader have two built-in annotation classes. Plug-ins can add movie, widget (form field), and other annotation types. You can define new annotation subtypes by creating new annotation handlers. For more information, see the *Acrobat and PDF Library API Reference.*

The IAC API has an object you can use to set, modify, and access annotation information from external applications. For more information, see *Interapplication Communication Developer Guide*.

New annotation types

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

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

To add a new annotation type, a plug-in must provide a set of callbacks, specify them in the appropriate structure, and register them. For more information, see the *Acrobat and PDF Library API Reference*.

1 XML and the Acrobat SDK

The Adobe XML architecture offers enterprises a step-by-step migration from manual, paper-based workflows to streamlined, automated processes that accelerate the flow of business-critical information between employees, customers, and suppliers. This chapter summarizes the Acrobat DC XML support and features.

Adobe XML architecture

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

XML forms model

The Adobe XML forms model uses a Document Object Model (DOM) architecture to manage the components that comprise a form. These include the base template, the form itself, and the data contained within the form fields. In addition, all calculations, validations, and formatting are specified and managed within the DOM and XML processes. Static are supported by Acrobat DC. A static form presents a fixed set of text, graphics, and field areas at all times.

XML templates

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

Extensible Metadata Platform (XMP)

Adobe's Extensible Metadata Platform (XMP) is a labeling technology that allows you to embed metadata into the file itself. With XMP, desktop applications and back-end publishing systems use a common method for capturing, sharing, and leveraging metadata. For more information, see "Metadata, Accessibility, and PDF Layers" on page 40.

SOAP and web services

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

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

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

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

For more information, see *Acrobat JavaScript Developer Guide*.

Conversion of PDF documents to XML format

Because XML representation is the basis for the exchange of information with web services and enterprise infrastructures, it is often useful to convert your PDF documents into XML format. It is a straightforward process to do this using JavaScript. For more information, see *Acrobat JavaScript Developer Guide*.

Alternatively, you can use the SaveAsXML plug-in, which extends the Save As Type choices in the Save As dialog box to allow a Tagged PDF document to be saved in a number of XML, HTML, or similar text-based formats. The plug-in uses mapping tables to control the conversion process for the SaveAsXML feature. For more information, see the *Acrobat Plugin Developer Guide*.

XML-based information

JavaScript provides support for XML-based information generated within workflows. For more information, see *Acrobat JavaScript Developer Guide*.

1 Forms and the Acrobat SDK

You can use the Acrobat DC SDK to extend the functionality of forms. Your form design can have dynamically changing features such as the current date, as well as convenience options such as automatic generation of e-mail messages. It can even have a dynamically changing appearance and layout that is responsive to user interactions.

Acroforms can be used to retrieve form data using JavaScript or using the Forms API with plug-ins and external applications. With Acrobat DC forms, you can author form fields and retrieve data from those form fields. For Acrobat Reader, the Forms plug-in does not allow form authoring, but allows users to fill in data and print Acrobat DC forms. The Acrobat Reader Forms plug-in also does not allow users to save data to the local hard disk. Both Acrobat DC and Acrobat Reader allow web designers to send data from the form back to a web server. For new forms development, use XML forms instead.

Workflows for forms

There are three basic workflows for forms:

- Forms are filled in on the screen and then printed out. They are then returned by traditional methods such as fax or postal mail.
- Forms contain a Submit button that enables the sending of an email message with an attached data file that contains only the form data.
- Forms submit form data to a web server much like forms on the Internet. The user must be online to submit the information.

About XML forms

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

Form population — You can populate forms from a database or using a standards-based network protocol such as SOAP. XML forms are particularly well suited for populating forms from an external database. For more information, see "SOAP and web services" on page 36.

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

Data collection — You can save your form data in pure XML format or save your forms in the XML Data Package format (XDP). The XDP format allow you to package units of PDF content within a surrounding XML container, and is thus considered an XML-based companion to PDF. The advantage of this format is that XML parsers provide direct access to the XML form-data subassembly of the PDF document.

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

Once you've collected PDF form data in FDF or XML format from multiple users, you can organize the form data into a comma-delimited spreadsheet file (CSV). After exporting the form data to a CSV file, you can work with the data in a spreadsheet application, such as Microsoft Excel. You can also save form data as a tab-delimited file. Tab-delimited files can be imported where required.

Data from various attachments can also be imported into an XML document and submitted to a server for processing.

About Acrobat forms

For Acrobat DC forms, a rich set of APIs allows you to create and manipulate form fields, and to retrieve form data using JavaScript, a plug-in, or an external application. Though you can manipulate form fields and form data from a plug-in, it is much quicker to develop Acrobat DC forms using JavaScript. Using the Acrobat DC SDK, you can perform the following tasks:

- Create Adobe PDF forms from scratch or from a template
- Add or remove form fields
- Set form field properties
- Make forms web-ready
- Extract and export form data
- Make forms accessible

You can extend the functionality of Acrobat DC forms with JavaScript. For example, you can use JavaScript to do the following tasks:

- Automate formatting, calculations, and data validation
- Develop customized actions assigned to user events
- Interact with databases and web services

Forms API

The Forms plug-in for Acrobat DC allows plug-in developers to author Acrobat DC form fields.

For Acrobat Reader, the Forms plug-in does not allow form authoring, but allows users to fill in data and print Acrobat DC forms. In general, the Acrobat Reader Forms plug-in does not allow users to save data to the local hard disk. However, if the PDF document has additional usage rights, then it may be able to save the document or perform other functions. For more information, see “Rights-Enabled PDF Documents and Security” on page 42.

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

OLE automation

The Acrobat DC Forms plug-in works as an automation server on Windows. There is no equivalent support on Mac OS. OLE automation is particularly useful for creating custom forms from an external application. Methods and properties are provided that allow you to programmatically associate actions and JavaScript with form fields. You can also add document-level JavaScript. For more information, see the *Acrobat and PDF Library API Reference*.

1 Metadata, Accessibility, and PDF Layers

Metadata

A PDF document can include general information such as the document's title, author, and creation and modification dates. Such global information about the document itself (as opposed to its content or structure) is called metadata, and is intended to assist in cataloguing and searching for documents in external databases.

Metadata properties and values are represented in the World Wide Web Consortium's Resource Definition Format (RDF), which is a standard metadata format based on XML. The set of standard metadata items is organized into schemas, each of which represents a set of properties from a particular application or industry standard. The schemas, as well as the physical representation, are defined by Adobe's Extensible Metadata Platform (XMP). For more information, see the *PDF Reference*.

Extensible Metadata Platform (XMP)

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

You can use JavaScript to access the XMP metadata embedded in a PDF document, and you can search metadata by using JavaScript or a plug-in.

Plug-ins can also get or set XMP metadata. Furthermore, you can get or set the XMP metadata associated with an internal PDF dictionary or stream. For more information, see the *Acrobat and PDF Library API Reference*.

Acrobat Distiller also supports embedding of XMP metadata in PDF files.

Adobe XMP Toolkit

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

Accessibility

Acrobat DC and the Acrobat DC SDK provide extensive support for accessibility of documents to visually-impaired users. To enable proper vocalization, either through a screen reader or by some more direct invocation of a text-to-speech engine, PDF supports the following features:

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

The core of this support lies in the ability to determine the logical order of content in a PDF document, independently of the content's appearance or layout, through logical structure and tagged PDF documents. Tagged PDF documents are created using the Acrobat Professional DC user interface or using the Acrobat DC SDK.

If a PDF document is untagged, you can convert it to a tagged PDF document using Acrobat Professional DC. Acrobat DC Standard provides only minimal support for tagging and no way to review or repair accessibility problems.

An accessible application can extract the content of a document for presentation to a user who is disabled by traversing the structure hierarchy and presenting the contents of each node. For this reason, producers of PDF files must ensure that all information in a document is reachable through the structure hierarchy.

For more information on accessibility support in PDF files, see the *PDF Reference*. For more information on implementing accessibility support, see the Acrobat DC Help and the *Acrobat JavaScript Developer Guide*.

PDF layers

Adobe PDF layers are components of content that can occupy the same physical space as other components. Multiple components may be visible or invisible depending on their settings, and may be used to support the display, navigation, and printing of layered PDF content by various applications.

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

Acrobat DC supports the display, navigation, and printing of layered Adobe PDF content output by applications such as Adobe InDesign®, AutoCAD, and Microsoft Visio. PDF layers are supported through the usage of PDF Optional Content Group (OCG) objects. Optional content refers to content in a PDF document that can be selectively viewed or hidden.

Note the following:

- You can rename and merge layers, change the properties of layers, and add actions to layers. You can also lock layers to prevent them from being hidden.
- You can control the display of layers using the default and initial state settings. For example, if your document contains a copyright notice, you can easily hide the layer whenever the document is displayed onscreen but ensure that the layer always prints.
- Acrobat DC does not allow you to author layers that change visibility according to the zoom level, but it does support this capability from other authoring applications.

- To direct users to a custom view using a particular layer set, you can add bookmarks to a PDF document that contains layers. Use this technique to highlight a portion of a layer that is especially important. You can add links so that users can click on a visible or invisible link to navigate to or zoom in on a layer.
- To create layers while exporting InDesign CS or later documents to PDF, make sure that Compatibility is set to the latest Acrobat DC version and that Create Acrobat Layers is selected in the Export PDF dialog box.

Creation of layered PDF files

When creating PDFs, several engineering applications, including Microsoft Visio and AutoCad, automatically generate the necessary ProcSets to create layered PDF documents.

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

Third-party developers must insert OC ProcSet information into the PostScript stream. For more information, see “Creating PDF files from an authoring application” on page 24 and the *Acrobat Distiller API Reference*.

What you can do with layers

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

Using JavaScript, you can determine the order in which layers are displayed in the user interface. You can also use JavaScript to perform the following tasks:

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

For more information, see *Acrobat JavaScript Developer Guide*.

For plug-ins, you use an object to represent an optional-content group. This corresponds to a PDF dictionary representing a collection of graphic objects that can be made visible or invisible. Any graphic content of the PDF can be made optional, including page contents, XObjects, and annotations. From a plug-in, you can perform the following tasks:

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

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

1 Searching and Indexing

With Acrobat DC and the Acrobat DC SDK, you can use the Search plug-in to locate an occurrence of a word in document content, metadata, attachments or other objects. You can use the Catalog plug-in to create a full-text index of a PDF document or document collection. This chapter summarizes these plug-ins.

Search plug-in

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

- Initiate a search with options. You can pass query expressions and search settings (for example, whole words only, word stemming, case sensitive, and so forth) to the Acrobat DC search plug-in and initiate the search. Search results will be presented to the user.
- Control the list of indexes to search. The search interface allows searches to be performed on one or more of the available indexes. You can control the list of active indexes.

The Acrobat DC Search plug-in allows users to perform text searches in PDF documents. It adds menus, menu items, toolbar buttons, and a Search panel to Acrobat DC or Acrobat Reader.

You can communicate with the Search plug-in through its plug-in API, IAC (DDE or Apple events) or JavaScript. You can perform all search options from each of these development environments.

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

Indexes and the Catalog plug-in

You can use the Acrobat DC SDK to create a full-text index of a set of PDF documents. A full-text index is a searchable database of all the text in the documents. After building an index, you can use search the entire library quickly. You can build and manipulate indexes from a plug-in, through JavaScript, or from an external application using IAC (DDE or Apple events) calls.

For indexing PDF files, Acrobat DC provides text extraction APIs. Text extraction also supplies position information that can be used to highlight search hits in the original PDF file. The text extraction tools are provided as calls in the plug-in API on Windows and Mac OS. You can extract ASCII text from a PDF file using a plug-in or using JavaScript. You can also save the PDF document as text or rich text.

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

For more information, see *Acrobat Plugin Developer Guide*, *Acrobat and PDF Library API Reference*, and the *Acrobat JavaScript Developer Guide*.

1 Developer Resources

Developer support

The Acrobat Developer Support team supports software development with the Acrobat DC SDK's core APIs. Developers can purchase support via the Adobe Creative Cloud Exchange Developer Support program. Supported SDK development activities include those for which the product is designed, tested, and licensed. Acrobat Developer Support does not support use cases that do not involve the Acrobat core API.

Only the last two major SDK versions with interim updates are eligible for support.

Note: For non-programmatic issues, such as questions about installing, using, customizing, or deploying Acrobat, contact Acrobat Technical Support

Licensing and distribution

Some of the Acrobat products may be licensed for commercial redistribution.

Products built around Acrobat DC products must comply with the Acrobat End-User License Agreement (EULA). For example, customers who purchase third-party developer products that use Acrobat must still comply with the Acrobat EULA.

Acrobat Reader

You may distribute Acrobat Reader, subject to the terms set forth in the license agreement. However, you must use the installer that comes with Acrobat Reader. You may invoke the Acrobat Reader installer from your installer, but the Acrobat Reader installer must remain intact and the End User License Agreement (EULA) must be displayed at first use. For information on customizing the Acrobat Reader installer, see How can I customize the Acrobat installer?

If you would like to distribute Acrobat Reader, see the Acrobat Reader distribution page given below in Additional resources.

Additional resources

Acrobat Reader distribution: http://www.adobe.com/products/acrobat/distribute.html?readstep

Volume software licensing: http://www.adobe.com/aboutadobe/openoptions/main.html

What are the technical and licensing limitations

The following is a list of Acrobat DC SDK uses that are not supported. Most activities are identified as *technically infeasible* and/or *contrary to licensing*.

- Use of any Acrobat product in a multithreaded way (*technically infeasible*).

  Any multithreaded access to the Acrobat core API is likely to crash or hang the application. Acrobat makes the following methods available that can help a plug-in or application manage its thread's access to the Acrobat core API, as documented in the *Acrobat and PDF Library API Reference*:

  ```
  AVAppRegisterNotification
  AVAppRegisterIdleProc
  ```

  When registering for a notification, the method is passed a function to be called by the Acrobat viewer when the event occurs. Registering for an `IdleProc` calls the function when nothing else is occurring. All notifications and `IdleProcs` are queued and called in order. Registering for notifications of events or `IdleProcs` can help a multithreaded application ensure that Acrobat is not accessed by multiple threads simultaneously; however, it is still the application's responsibility to manage its threads that access Acrobat.

- Use of any Acrobat product as a server process accessed by multiple clients, unless it is specifically stated in product documentation as designed and licensed for such purpose (*technically infeasible | contrary to licensing*).

  Unless specifically stated in the product documentation and licensing agreement, Acrobat products are not designed, tested, or licensed for this purpose. This use is in violation of the licensing restrictions, as described in the End User License Agreement displayed during Acrobat installation and is not supported by Acrobat Developer Support.

- Use of Distiller as a Windows NT service (*technically infeasible*).

  Acrobat Distiller requires the ability to open a window on the desktop to run. It is not possible to use Distiller as a Windows NT service.

  Note: For customers needing this capability, Acrobat Professional DC 7 introduced support for *watched folders*. This same support is also present in Acrobat Pro Extended 10. When Distiller finds a PostScript file located in the In folder of a watched folder, it converts the file to PDF and then moves the PDF (and usually the PostScript file and any associated log file) to the Out folder.

- Use of `MenuItemExecute` to bring up Acrobat DC dialog boxes when a PDF file is displayed in an external window using OLE automation (*technically infeasible*).

  Due to problems of managing window focus, using the Acrobat DC dialog boxes (using `MenuItemExecute`) when a PDF file is displayed in an external window using OLE automation is not supported. The actions executed by the dialog box may or may not affect the intended PDF file and there can be problems of windows not redrawing properly. The only supported workaround is to use the OLE automation methods or to develop a plug-in to achieve any functionality not available in the OLE API to Acrobat DC. The Adobe Reader Integration Key License Agreement only permits displaying in an external window when Acrobat DC is used, not Acrobat Reader.

- Use of the *PDF Reference* to develop a third-party application that writes PDF files without the use of Acrobat DC products.

  Note: The Adobe PDF Library, available under license, can be used to simplify development of these types of applications.

  We do not provide support to developers who are creating their own PDF generation capabilities without the use of Adobe products. The *PDF Reference* is the best resource for this kind of development. The use of Adobe products to create PDF files as a benchmark for your own development is recommended. Acrobat Developer Support will not debug PDF files created with non-Adobe products. Questions regarding the completeness or accuracy of the *PDF Reference* will be answered. The *PDF Reference* is available for free download from the Acrobat Developer Center.

- Interapplication communication (IAC) between a plug-in and a third-party application.

  Interapplication communication between a plug-in and a third-party application does not differ significantly from interapplication communication between two stand-alone applications. Documentation for your development platform's API and development environment are the best resources for this type of development. The Acrobat DC SDK contains two samples that can serve as examples.

  - `DDEServer` demonstrates DDE communication between a stand-alone application and a plug-in.
  - `ExternalWindow` uses Windows messaging to communicate with a stand-alone application.

  One typical difficulty for developers occurs when a multithreaded stand-alone application communicates with a plug-in. See Developing for Acrobat Reader.

- Use of platform API methods or the API methods of applications other than Acrobat DC.

  Acrobat Developer Support can help you with the API to the Acrobat family of products. Questions regarding the use of platform API methods should be directed to the manufacturer of your operating system.

- Use of the ActiveX® control or Netscape plug-in to display a PDF file in an external application besides Internet Explorer or Netscape. The methods used by Acrobat DC to display a PDF file in Netscape and Internet Explorer are intended only for use with these browsers. Use of the ActiveX Control and Netscape plug-in installed by Acrobat Reader is not licensed to other applications. Development with these interfaces is not supported and no documentation is available.

- Automating the import of image files using the Import plug-in to Acrobat DC (*technically infeasible*).

  The Import plug-in to Acrobat does not provide an API that allows it to be called from a plug-in or another application. Executing the Import Image menu item with `MenuItemExecute` brings up a dialog box requiring user input.

How can I customize the Acrobat installer?

Adobe provides various ways in which you can deploy Acrobat products to a large number of systems.

You can find documentation about enterprise installation, including information about tools for customization and installation as well as guidelines for extending enterprise applications, in the Enterprise Toolkit.

1 Frequently Asked Questions

Forms

What are the requirements for using Acrobat forms?

You can create and fill in Acrobat forms in PDF files, and submit or import form data. These forms are typically used for viewing or printing information, filling in information, selecting options, digitally signing documents, and exchanging information with databases. Note that Reader users can fill in and submit forms. Once a user fills out an Acrobat form, the data can be submitted to a server for processing. Data can be exported from a PDF form into Forms Data Format (FDF) or XML-based FDF (XFDF). Data can be imported into a PDF form if it is in FDF, XFDF, XML, or TXT format. Acrobat forms support the following formats: FDF, XFDF, tab-delimited text, and XML.

Every FDF file contains a reference to a PDF file for which the data is intended, designated with the `/F` key inside the FDF file (unless the FDF file is for the same PDF from which you submitted your data). When Acrobat or Acrobat Reader receives an FDF file, it opens the corresponding PDF file, and fills the form fields with the data from the FDF file. If the PDF file is referenced by a URL (for example, http://example.com/file.pdf), the FDF file must be sent to the server in response to a `submit` action from a PDF form. For more information about the FDF format, which is based on PDF, see the *PDF Reference*.

Acrobat plug-ins can programmatically import FDF data into a PDF file from a local file system using the HFT made available by the Forms plug-in. OLE automation can be used to programmatically add, modify, or delete form fields, import or export FDF data, execute scripts written in JavaScript, and much more. For more information, see the *Acrobat and PDF Library API Reference.*

What is the FDF Toolkit?

The FDF (Forms Data Format) Toolkit is a thread-safe API for writing a server application that generates FDF data or parses FDF data from a form created by the Acrobat Forms plug-in. If you need to parse FDF files submitted from a PDF form, or generate FDF files to be submitted to a PDF form, you can use the FDF Toolkit. The FDF Toolkit supports COM on Windows, C and Perl on Windows, Solaris™, AIX® or Linux®, and Java VMs compatible with versions 1.2 or later.

For detailed information, see the documentation included with the FDF Toolkit.

PDF documents

What ActiveX solutions are available?

The PDF browser control automation object, described in the *Interapplication Communication Developer Guide*, supports the loading and display of PDF documents as ActiveX documents, and its interface is available in both Acrobat and Acrobat Reader.

Visual Basic .NET and Visual C# .NET

How can I use Visual Basic .NET or Visual C# .NET?

The Windows version of Acrobat is an OLE automation server. In order to use the OLE objects made available by Acrobat, you must have the full Acrobat product installed on your system and the `Acrobat.tlb` file included in the project references for your Visual Basic project. This allows you to use the Visual Studio 2005 object browser to browse the OLE objects.

For Acrobat versions 5.0 and later, it is possible to access the JavaScript object model in Acrobat through the OLE automation `JSObject` interface. See the `JSObjectAccess` sample. For more information, see the *Interapplication Communication Developer Guide*.

What resources are available for Visual Basic .NET or Visual C# .NET?

The *Interapplication Communication Developer Guide* describes the objects and methods available in these languages, as well as guidelines for usage. These documents (as well as the API) were designed with C programming in mind, and programming with the API requires some familiarity with C concepts such as `enum`.

Besides the object browser, the best resources for programming in these languages are the sample projects. The samples demonstrate use of the Acrobat OLE objects and contain comments describing the parameters for the more complicated methods. For more information, see the *Acrobat SDK Samples Guide*.

The `Iac.bas` file in the InterAppCommunicationSupport\Headers folder contains the Visual Basic counterparts to the C `#define` statements and enumerations passed to many of the API methods.

What API methods are available to modify PDF documents?

Modifying the page contents of a PDF file is primarily accomplished using the Acrobat core API.

Using the Acrobat core API greatly simplifies modifying and creating PDF page contents. To demonstrate this, there are several snippets available from the `SnippetRunner` sample plug-in that show you how to add data to the contents of a page, while ensuring that the PDF file is still readable after modification. To attempt to do this without using the core API would be significantly more difficult and could result in an unreadable PDF file.

Acrobat’s automation interfaces are limited mainly to what a user can do through the user interface and cannot modify the contents of a page.

For more information on the Acrobat core API, see *Acrobat Plugin Developer Guide* and the *Acrobat and PDF Library API Reference*.

Can I modify PDFs without a C programming background?

You can make some modifications to PDF files through JavaScript for Acrobat. See the *Acrobat JavaScript Developer Guide*.

How can I extract text?

You can extract text with the Acrobat DC SDK in two ways:

- Use the Acrobat core API
- Use the Acrobat automation API

Through the core API, you can extract ASCII text from a PDF file using Acrobat and a plug-in developed in C or C++. The `TextExtraction` and `WordFinder` sample plug-ins demonstrate text extraction and can be used as starting points for your own plug-in. `AVConversion` methods also can be used to save PDF as text or rich text. In addition, the `SDKTextExtraction` sample in the first level of the SaveAsXML directory provides a good starting point for creating richer extraction tables. For more information, see the *Acrobat SDK Samples Guide*.

How can I display a PDF in an external application window?

There are several ways to have the Acrobat program display a PDF file in an external application's window. Acrobat must be installed on the system to view a PDF file in your own application's window.

There is no best way that we suggest to display PDF files in your application. Examine the following list for the most appropriate method for your situation. For more documentation, see *IAC Developer Guide*.

Using Acrobat to view a PDF file in your own application's window

Windows only:

- OLE automation, using the `OpenInWindowEx` command. This displays a live view of the PDF file in the OLE application window.

  For samples, see the Visual Basic and C++ `ActiveView` applications.
- OLE automation, using the `DrawEx` command. This displays a bitmap of the current page in the OLE application window.

  For samples, see the Visual Basic and C++ `StaticView` applications.
- Copy to the clipboard. This copies a PDF image to the clipboard without requiring an `hWnd` or `hDC` using OLE automation.
- If you are using simplified browser controls in your application, you may treat the PDF document as an ActiveX document by using the `AcroPDF` object's `LoadFile` method. This automation interface is available in both Acrobat and Acrobat Reader.

Mac OS only:

- The `AcroAppleEvents` sample application demonstrates rendering a PDF file into another application's window (see `DrawIntoWindowCommand` for details).

Windows and Mac OS:

- The `ExternalWindow` sample plug-in demonstrates a live view of the PDF file in a window created by the plug-in. You can extend this to display PDF files in an external application window.

Are multibyte font PDF documents supported by the Acrobat SDK?

Acrobat allows the creation, modification, and use of PDF files containing multibyte fonts.

How are security and encryption provided in PDFs?

Adobe provides password-based and certificate-based encryption schemes with Acrobat. In the password-based scheme, Acrobat queries the user for a password before a file is opened. The authorization procedure to open a file or to set an owner password can be modified by creating a custom security handler.

Full-text search

What tools are available with Acrobat for full-text search?

Adobe provides a full-text search system, known as the Acrobat Search Plug-in, based on third-party technology. However, this does not preclude other search systems from integrating with Acrobat. The Acrobat search system was created using only public API and IAC calls, and you can easily remove it from Acrobat and replace it with another search technology. For more information, see the *Interapplication Communication Developer Guide*.

The Acrobat interface can be extended with new menus and toolbar icons to allow tight integration with your search plug-in and Acrobat. For example, buttons to invoke a search and to find the next or previous occurrences can be added to the toolbar.

For samples of adding menu items and toolbar buttons, see the related snippets in the `SnippetRunner` sample plug-in.

What tools are available for extracting and highlighting text?

For indexing and searching PDF files directly, Acrobat provides support through IAC and plug-in calls. For indexing PDF files, Acrobat provides text extraction APIs. For further information on extracting text from PDF, see How can I extract text?

How do I communicate with the Acrobat Search plug-in?

You can communicate with the Search plug-in via its plug-in API or via IAC (DDE or Apple events). Using either of these methods, you can control the Acrobat Search plug-in in the following ways:

- Control the list of indexes to search.

  The search interface allows for searches to be performed on one or more of the available indexes. You can control the list of active indexes.
- Initiate a search with options.

  You can pass query expressions and search settings to the Acrobat search plug-in and initiate the search, and the results will be presented to the user.

For an example, see the `SearchPDF` Visual Basic application. For documentation, see the *Acrobat and PDF Library API Reference.*

How do I create custom DocInfo fields for searching?

The Catalog feature is used to create a search index; its API is available to third parties via the Catalog plug-in provided with the Acrobat DC SDK. See the *Acrobat and PDF Library API Reference* and the *Interapplication Communication Developer Guide* for more information.

Once you have built the search index, you must create the custom `DocInfo` fields, which can be used to search document metadata. You can accomplish this task with either the Acrobat core API or using interapplication communication. See theadobefor more information.

At this point you may submit custom queries to the Acrobat Search plug-in to search by the custom fields, as demonstrated by the `SearchQuery` sample plug-in. You can include your custom `DocInfo` fields in the search by specifying them in the `sortSpec` parameter of the `SearchExecuteQuery` method.

How do I use the Windows command line?

You can display and print a PDF file with Acrobat and Acrobat Reader from the command line. These commands are unsupported, but have worked for some developers. There is no documentation for these commands other than what is listed below.

Note: All examples below use Acrobat Reader, but apply to Acrobat as well. If you are using Acrobat, substitute `Acrobat.exe` for `AcroRd32.exe` on the command line.

`AcroRd32.exe` ***`pathname`*** — Start Acrobat Reader and display the file. The full path must be provided.

This command can accept the following options.

Option	Meaning
`/n`	Start a separate instance of Acrobat or Acrobat Reader, even if one is currently open.
`/s`	Suppress the splash screen.
`/o`	Suppress the open file dialog box.
`/h`	Start Acrobat or Acrobat Reader in a minimized window.

`AcroRd32.exe /p` ***`pathname`*** — Start Acrobat Reader and display the Print dialog box.

`AcroRd32.exe /t path` ***`"printername" "drivername" "portname"`*** — Start Acrobat Reader and print a file while suppressing the Print dialog box. The path must be fully specified.

The four parameters of the `/t` option evaluate to `path`, `printername`, `drivername`, and `portname` (all strings).

`printername` — The name of your printer.

`drivername` — Your printer driver's name, as it appears in your printer's properties.

`portname` — The printer's port. `portname` cannot contain any "/" characters; if it does, output is routed to the default port for that printer.

Index