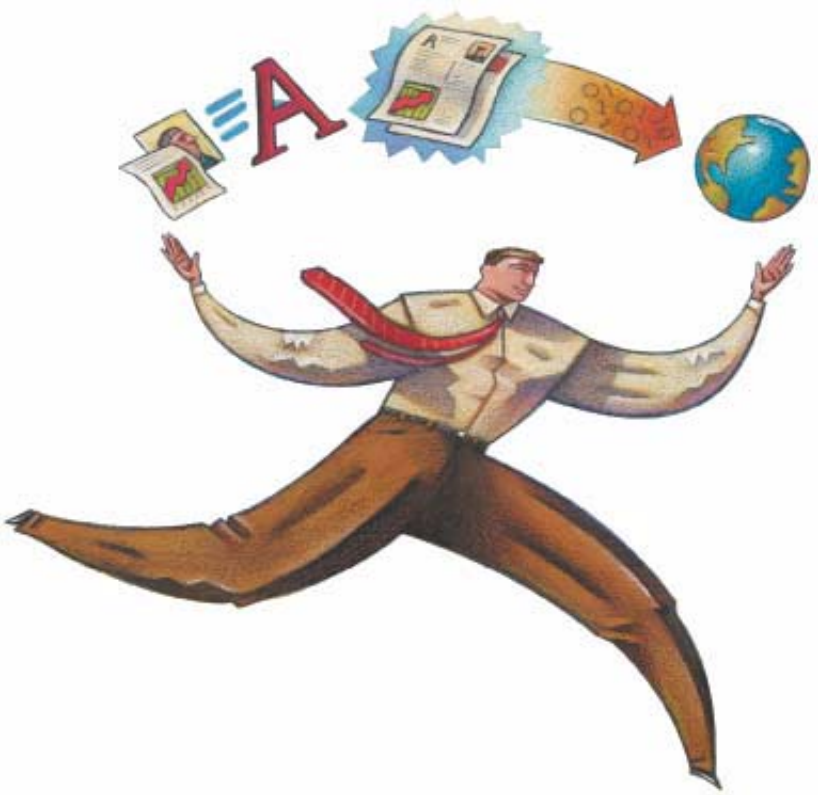
Acrobat Digital Signature Overview

Technical Note #5400

Version :Acrobat 5.0



ADOBE SYSTEMS INCORPORATED

Corporate Headquarters

345 Park Avenue

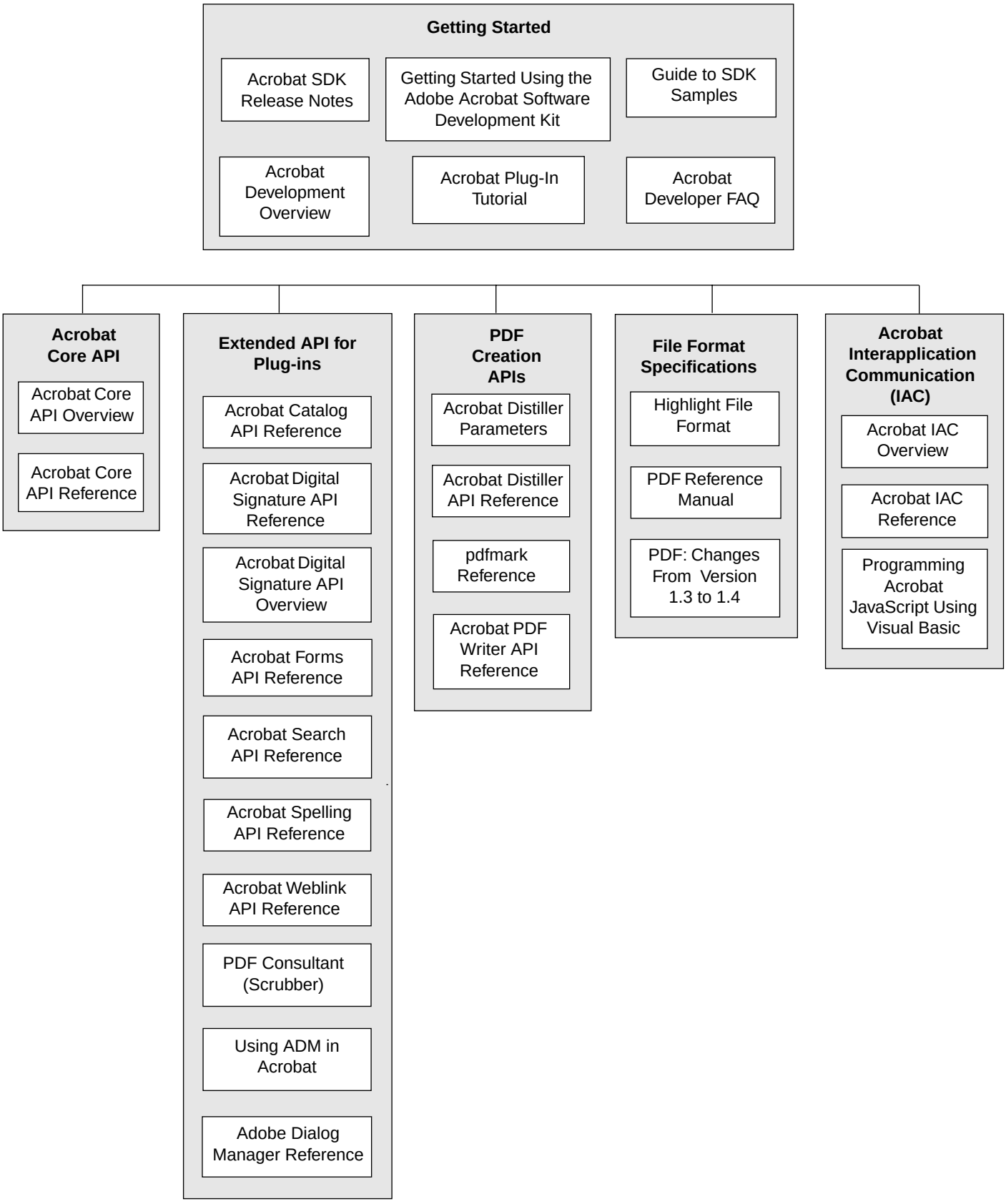
San Jose, CA 95110-2704

(408) 536-6000

http://partners.adobe.com

April 2, 2001

Acrobat SDK Documentation Roadmap



Contents

Contents

Preface

People have traditionally used signatures as a means of informing others that the signator has read and understood a document. Now with Acrobat's Digital Signature plug-in (DigSig) people can digitally sign a document for the same purposes. An Acrobat signature in a document is bound to that document in such a way that altering the signed document or moving the signature to a different document invalidates the signature. This security eliminates the need for paper copies of documents and can speed the processes involving documents that require signatures.

A single document may be signed more than once, and changes may occur between signings. Acrobat's Digital Signatures links each signature with a particular state of the document. All changes append the PDF changes to the fully-preserved base PDF document. The ability to do serial signatures of protected documents is unique to Acrobat, and draws heavily on the PDF file design for an appended save.

To support Digital Signature plug-ins, Acrobat supplies a series of callbacks. This document describes the Digital Signature callbacks and outlines the process to create a plug-in using a digital signature.

If you received this technical note without obtaining the entire Acrobat Software Development Kit (SDK), you can get the complete SDK by visiting:

http://partners.adobe.com/asn/developer/acrosdk/main.html

What Is in This Document

This document explains the concepts of a digital signature and takes you through some suggested steps in creating plug-ins that work with Acrobat's Digital Signatures plug-in.

Who Should Read This Document

You should read this document if you are a security developer who wants to extend your security measures to your customers' PDF documents. This document assumes that you are writing plug-ins to customize Acrobat's Digital Signature capabilities to your customers' needs.

Prerequisites

You should already be familiar with the process for creating Acrobat plug-ins. If you are not, please read *Acrobat Plug-In Tutorial* first.

You should already be familiar with the process of creating annotation appearances and Form XObjects. Refer to the *PDF Reference* for more inforamtion.

Other Useful Documentation

The Acrobat SDK includes many other books that you might find useful. When mentioned in this document, those books will often appear as live links (blue italic links). However, in order to actually jump from this document to those books, those books must exist in the proper directories within your computer's file system. This happens automatically when you install the SDK onto your system.

If for some reason you did not install the entire SDK onto your system and you do not have all of the documentation, please visit the Adobe Solutions Network web site (http://partners.adobe.com/asn) to find the books you need. Then download them and install them in the proper directories, which can be determined by looking at the Acrobat SDK Documentation Roadmap, included at the beginning of each book in the SDK.

You should be familiar with the Acrobat core API and Portable Document Format (PDF). The following technical notes provide this information.

Acrobat Digital Signature API Reference, Technical Note #5192. Provides a reference of the methods, objects, declarations and callbacks that make up the Digital Signatures API.

Acrobat Core API Overview, *Technical Note #5190*. Gives an overview of the objects and methods provided by the Acrobat core API.

Acrobat Core API Reference, Technical Note #5191. Describes in detail the objects and methods provided by the Acrobat core API.

PDF Reference Manual, Second Edition, version 1.3. Provides a description of the PDF file format, as well as suggestions for producing efficient PDF files. It is intended for application developers who wish to produce PDF files directly.

Conventions Used in This Book

Acrobat documentation uses certain text styles to identify various operators, keywords, terms, and objectss

TABLE P.1 Acrobat SDK document conventions

Item	Character Definition	Use and Examples
File names	Courier, 12-point	`C:\templates\Acrobat_docs`
Code items within plain text; parameter names in reference documents	Courier, 12-point, bold	The **`GetExtensionID`** method returns an **`ASAtom`** object
Code examples set off from plain text	Courier, 10-point, plain	These are variable declarations: `AVMenu commandMenu,helpMenu;`

Pseudocode	Helvetica, 11-point, italic	`ACCB1 void ACCB2 ExeProc(void)`
		`{`
		`do something`
		`}`

Cross references to Web pages	Blue text; everything else “as-is”	The Acrobat Solutions Network URL is: partners.adobe.com/asn/
Cross references to titles of other Acrobat SDK documents	Blue text; Helvetica, 11-point, italic	See the *Acrobat Core API Overview*.
Cross references within a document	Blue text; everything else “as-is”	See Section 3.1, “Using the SDK.” Test whether an **`ASAtom`** exists.
PostScript language operators, PDF operators, keywords, dictionary key names;user interface names	Helvetica, 11-point, bold	The **setpagedevice** operator The **File** menu
Document titles that are not cross-reference links, new terms, PostScript variables	Helvetica, 11-point, italic	*Acrobat Core API Overview* *filename* **deletefile**

1 Digital Signature Overview

Introduction

When Acrobat launches, all plug-ins go through a three-step initialization process:

1. Export Host Function Tables (*HFT*s).
2. Import HFTs.
3. Perform initialization.

This sequence allows plug-ins to establish communication among themselves without being dependent on the order of loading. For digital signatures, the sequence is:

1. Acrobat's Digital Signature plug-in (*DigSig*) exports its HFT under the name **`DigSigHFT`**.
2. Your plug-in imports the DigSig HFT.
3. Your plug-in calls **`DigSigRegisterFilter`**, which allows you to register all their procedures into **`DigSigHandlerRec`**'s structure. Later, DigSig will consult this structure to look up your appropriate procedures to use at callback times.

Eventually, the user opens a document. DigSig calls your plug-ins to notify them of the new document by calling **`DSDocOpenProc`**; you might allocate some storage or choose to automatically validate any of their respective signatures in the document.

Auto-validation may encounter significant delays if it requires reading the entirety of a large document off a CD-ROM or over a network, or if it requires accessing some signature registry or authority over a network; Adobe software will only access signatures at user request.

Eventually, the user closes a document. DigSig calls **`DSDocCloseProc`**.

Digital Signature Components

Digital signatures are composed of two parts:

- The signature field dictionary: the PDF dictionary structure that stores information about the signature
- (Unless it is a blind digital signature) the signature annotation with its associated appearance: the appearance of the signature, including the background, and layout of name, time, etc.

Acrobat's Digital Signature plug-in creates these two parts when the user chooses to sign a document. Your plug-ins do not have to handle deleting the signature, as the DigSig plug-in does that transparently.

Digital Signature Scenarios

Acrobat supports three digital signature scenarios. Acrobat's Digital Signature plug-in handles the first two cases; you as a developer will be interested in the second and the third.

1. If the user creates a signature field and does not specify a default signing method, DigSig handles that case with no communication to your plug-ins:
   - DigSig creates the signature field dictionary.
   - DigSig creates the signature annotation dictionary.
   - DigSig creates the (blank) signature appearance dictionary.
2. The Forms plug-in also creates Signature fields. If the user creates a signature field and specifies a default method, Forms calls DigSig to fill in default values:
   - DigSig creates the signature field dictionary.
   - DigSig creates the signature annotation dictionary.
   - DigSig creates the (blank) signature appearance dictionary.
   - DigSig calls the **`DSDefaultValueProc`** callback that your plugin provides. This callback must create the default signature value dictionary and create the **/DV** key in the signature field dictionary to point to it.
3. If the user asks to sign a specific signature field using the plug-in, DigSig sequences the interaction. This sequence is a four-step process consisting of callbacks into your plug-in. Your plug-in needs to register these callbacks during the plug-in initialization phase by creating a **`DigSigHandlerRec`** structure, assigning the relevant methods, and then calling **`DigSigRegisterFilter`** to register the structure. The four necessary callbacks are **`dsNewSigData`**, **`dsCommitSign`**, **`dsFinshSign`**, and **`dsFreeSigData`**. All four callbacks need to be assigned methods for the callback process to occur.

Understanding the Process

The steps in this section are suggestions that describe the interactions of a digital signature plug-in. (The **`SignDoc`** sample plug-in provided with this SDK is a more complete example.)

Dialogs and Signature Gathering

1. DigSig calls your **`dsNewSigDataProc`**, a callback that begins the process.

- Your plug-in interacts with the user, and allows the user to cancel if they want to do so.
- Your plug-in acquires the signature itself in a method-specific way. All information is saved in memory, without altering the document itself. This allows a later backout.
- If **dsNewSigData** does not cancel, DigSig prepares the document for saving: It first calls **dsUnValidateSig** on every signature in the document to put any overprinting/underprinting in canonical form. It then counts how many pages and fields have changed since any prior signature and records this.
- For a first signature, DigSig does the SaveAs dialog, allowing the user to select filename, optimization, and encryption. The user may cancel. Other than fatal errors, such as out-of-disk-space, this is the last chance to stop the process.

Saving the Document

2. DigSig calls Xxxx.**DSCommitSignProc** to update the document with the actual signature.

   Your **DSCommitSignProc** callback must:
   - create the signature dictionary, possibly using information in the signature field **/DV** dictionary, perhaps using the **/ByteRange** and **/Contents** keys.
   - point **/V** in the signature field dictionary to this. Then create the **/AP /N** value in the signature annotation dictionary, using a method-specific visible representation of the signature, including a symbol signifying “unvalidated signature.”
   - optionally allocate dynamic storage for a marked array, an array of “marked” COS objects that it cares about.
   - return a marked array that includes at least the **/ByteRange** and **/Contents** value objects.

3. DigSig inserts the **/Changes** array from step 1.

4. DigSig saves the PDF document to a file. For each Cos object in the marked array, DigSig records the object’s byte offset and length in the file as written. The saved file may have objects encrypted by the Acrobat standard encryption handler, if the user so chooses.

5. The very first time a document is signed, DigSig may rename the file and may invoke the Optimizer, Linearizer, and Garbage Collector. Upon return from the save, all COS objects are invalid, including those in the marked array.
   - All PD-level objects except the **PDDoc** are invalid. Signing methods must not depend on saving any such state between **dsCommitSign** and **dsFinishSign**. In particular, the byte offsets and lengths in the marked array are valid upon entry to **doSign**, but the Cos objects are not. The order of entries is unchanged, however, these Cos objects will be rewritten by DigSig as **CosNull** before calling **dsFinishSign**.

Finishing the Process

6. DigSig calls **dsFinishSign**, passing back in the marked array.

   Your **DSFinishSignProc** must:
   - Calculate the **/ByteRange** that it desires, using the byte offsets and lengths in the marked array.
   - Overwrite the marked **/ByteRange** value in the saved file, using the **DigSigOverwriteIntArray** or **DigSigOverwriteBytes** callback.
   - Overwrite any other marked Cos objects it wants to.
   - Calculate any document digest that it desires, using the **DigSigFileGetEOF**, **DigSigFileSetPos**, and **DigSigFileRead** callbacks; or it may use the **DigSigMD5ByteRange** callback.
   - Obscure or encrypt this digest in a method-specific way.
   - Overwrite the marked **/Contents** value in the saved file, using **DigSigOverwriteHexstring** or **DigSigOverwriteBytes**.
   - Optionally delete dynamic storage for the marked array the plug-in returns.

7. DigSig calls **dsFreeSigData**, which may free up any remaining storage.

Revalidating Signatures

If the user reopens the file, the signatures must be revalidated. If the user asks to validate one or more signature fields, DigSig sequences through them one at a time:

1. DigSig calls **validateSign**. Your **DSValidateSignProc** must:
   - Recalculate any document digest that it desires, using the **DigSigFileGetEOF**, **DigSigFileSetPos**, and **DigSigFileRead** callbacks; or it may use the **DigSigMD5ByteRange** callback.
   - Compar this result to the stored one, and do any other method-specific checks it desires.
   - Optionally do a validation against some stored (network) registry.
   - Update the **/AP /N** value in the signature annotation dictionary to show **doublechecked/pass/fail** symbol.
   - Return **doublechecked/pass/fail**.

The user may open more than one document at a time, and may switch between open documents at will.

Additional Available Callbacks

The user may ask to show a signature panel containing summary information for each signature in an open document. If multiple documents are open, there may be multiple panels, or a single panel may be repainted as the user switches between documents. DigSig manages updating the panel(s), but may call the respective

method plug-in for each signature to get information to display on the panel. For each signature, the signature panel has two levels of detail:

1. **CLOSED** displays a **doublechecked/pass/fail/unknown/blank** icon and a line of text for each signature field in the document. The default text is the name of the person signing and the date and time of signing, displayed in a language-independent way.
2. DigSig calls **dsGetValidState** to choose which icon to show.
3. **OPEN** displays an icon and line of text for each signature, then indented lines of further text, currently consisting of the name of the signer, date and time of signing, location of signing, reason for signing, and signing method.
4. DigSig calls **dsGetValidState** to choose which icon to show.

Your plug-in may update the signature panel for a document asynchronously (it might be doing validation as a background or idle-loop task). To do this, use the **DigSigUpdatePanel** callback.

Additional DigSig Plug-in Support

Whenever a signature is created or verified, the plug-in may optionally alter the appearance of the signature in the document, for the purpose of displaying or printing. For example, it could change an overprinted question mark on an unverified signature to an underprinted logo for a verified signature. To help with this, DigSig provides an HFT callback **DigSigGetStdXObj** that returns an **XObject** for a blank appearance, a question mark, or a cross. These are suitable as targets of the **Do** operator in a signature's appearance stream.

To avoid saving a signature to a file with an appearance of valid (rather than unvalidated), just before each file save, DigSig loops through all the signature fields and calls the specific method's **dsUnValidateSig** entry. This routine restores the signature's appearance to the unvalidated state.

The AcroForms Widget Annothandler calls into DigSig using four entries. These calls all reflect user actions taken in the document view, not the Signatures panel view.

When the user selects an annotation by tabbing to it or by clicking it with the mouse, and that annotation is for a signature field, AcroForms calls **DigSigDraw**. If the annotation is selected, then **bIsSelected** is **true**.

When the user tabs to a signature annotation and activates it by hitting the spacebar or enter key, this is equivalent to a left mouse click.

AcroForms calls **DigSigKeyDown**. The parameters parallel those of **AVAnnotHandlerDoKeyDownProc**.

When the user left-clicks inside a signature annotation, AcroForms calls **DigSigClick**. The parameters parallel those of **DoClickProcType**.

When the user right-clicks inside a signature annotation, AcroForms calls **DigSigRightClick**.

Rollback Support

There is a constraint on the values in the **/ByteRange** array. This constraint allows DigSig to implement rollback to prior signatures:

The largest offset + length value in the **/ByteRange** array for a given signature must be equal to the length of the PDF file containing that signature; that is, it must equal offset + 1 of the "F" in the **%%EOF** at the end of the file.

In addition, the following constraints also apply:

- All offsets must be in the range 0..2147483647
- All lengths must be in the range 1..2147483647
- Offset[n+1] must be strictly greater than offset[n] + length[n]

Interoperability With Public-Key Technology

You can use public-key technology to work with multiple files that have digital signatures and multiple file handlers. There are two Adobe documents you can read to find out more about this:

- Acrobat Public-Key Interoperability — provides an overview of the concepts of using pubic-key technology with Adobe digital signatures
- PDF Public-Key Digital Signature and Encryption Specification — provides the required standard for the signature and encryption dictionary formats.

You can find these documents at
http://partners.adobe.com/asn/developer/acrosdk/docs.html