Digital Signature Appearances

Version : Acrobat 6.0

ADOBE SYSTEMS INCORPORATED
Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000
http://partners.adobe.com

May 2003

Contents

Contents

1 Digital Signature Appearances

This document provides guidelines for the implementation of Signature Appearances in a PDF file. The document contains information that is relevant to third party developers as well as some general information that is relevant to system administrators.

Signature Interoperability

Acrobat's digital signature API allows plug-ins to implement *digital signature handlers* to create and verify document signatures.

An important goal is to achieve file interoperability between handlers. This is particularly relevant for handlers that are implemented based on public-key cryptography. This interoperability will allow a PDF file that is signed with one digital signature handler to be verified with a different digital signature handler, assuming the other handler understands the same basic signature format.

To achieve interoperability, a standard PDF syntax for implementation of signatures is required. There are two primary components to this standard syntax:

- *The signature dictionary*: where the actual signature is stored. It includes attributes such as the name of the signer, the time of the signature, the signed hash of the file, and the signer's certificate (e.g. embedded as a PKCS#7 object). Standard syntax for the signature dictionary is defined using the **SubFilter** attribute of the signature dictionary. This is described in version 1.5 of the *PDF Reference,* and previously in the technical note *PDF Public-Key Digital Signature and Encryption Specification*.
- *The signature appearance*, which is an optional appearance that is contained in the signature annotation. Standardizing the appearance is not necessary for the cryptographic purposes of verifying a signature. However, it does make for a consistent user experience. This doccument describes the techniques for standardizing signature appearances.

Annotation Appearances

Acrobat Digital Signatures exist as Signature Form Fields in a PDF file. As with any form field, signature form fields are always associated with an annotation dictionary. Appearances for any annotation, including signatures, are contained in the appearance dictionary, or **AP** attribute, of the annotation.

Digital Signature handlers are free to populate and manipulate the signature appearance as they see fit. If it is expected that other handlers be able to manipulate this appearance, there needs to be a standard by which this appearance is created. This document provides

guidelines for the generation of standardized appearances. The guidelines are described beginning in Standard AP Dictionary. Even if appearances are not meant to be manipulated, it is still recommended that third party developers follow the guidelines.

Adobe signature handlers allow appearances to be manipulated so that the validity state of the signature can be displayed as part of the appearance. Example states include *blank*, *unknown*, *valid* and *invalid*. When such a document is opened the validity of the signature can be programmatically determined, and then the appearance is updated to reflect the validity state.

Acrobat versions 4.0 through 5.05 generated signature appearances that displayed the validity state of a signature as part of the signature appearance. The appearance was initially saved in the unknown state, using a yellow question mark, and was updated whenever the validity of the signature was determined. These appearances conform to the guidelines described in Standard AP Dictionary

Beginning in Acrobat 6.0, signature appearances are no longer generated with the intent of incorporating the validity state as part of the signature appearance. The appearances generated in Acrobat 6.0 continue to conform to the guidelines described in Standard AP Dictionary, but do not use the layers that are reserved for validity display. Acrobat 6.0 continues to support legacy signatures that include validity information as part of their appearance, when these appearances conform to these guidelines. Specifically, Acrobat 6.0 continues to use the layer mechanism described in this document, however layers **n1**, **n3** and **n4** are not generated for new signatures.

Adobe recommends that signature handlers follow Adobe's lead by discontinuing generation of layer **n1**, **n3** and **n4** in new signatures. Adobe also recommends that plug-in developers make use of the complete signature appearance support that is now part of the PubSec layer of code that resides between the new Acrobat 6.0 PubSecHFT and the existing DigSigHFT. Refer to Acrobat 6.0 Appearance Management for further details on the Acrobat 6.0 implementation and customisation.

Example Signatures

These examples are referred to later in this document.

FIGURE 1.1 Unverified signature, created using PPKLite 4.0.

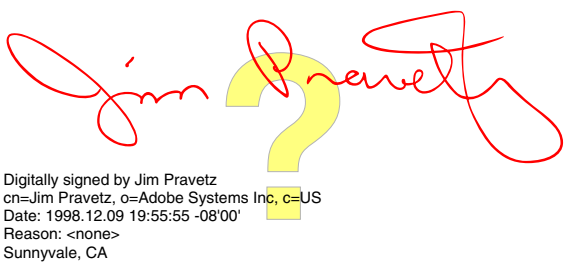


FIGURE 1.2 ***Invalid signature, created using PPKLite 4.0.***

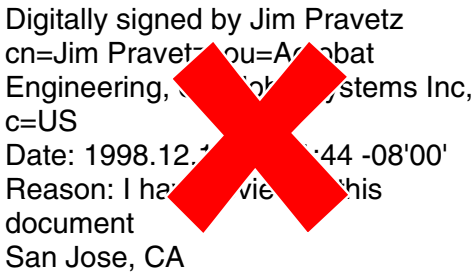


FIGURE 1.3 ***Unverified signature with n4 layer for status text, created using PPKLite 4.0.***

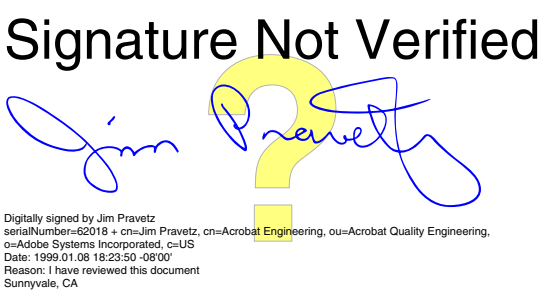


FIGURE 1.4 Signature appearance created using PPKEF 4.0

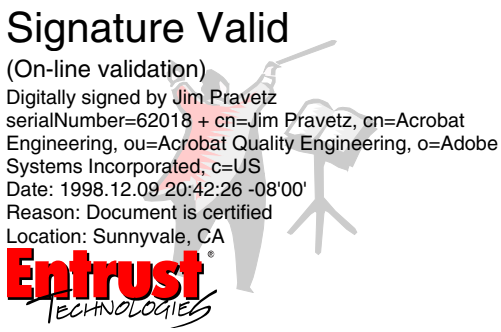
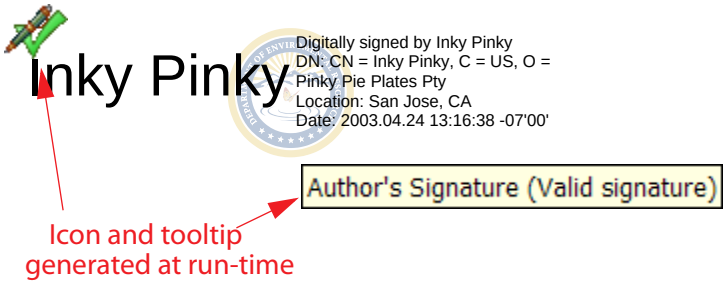


FIGURE 1.5 Signature Appearance created using Acrobat PubSec 6.0

FIGURE 1.6 Signature Appearance created using Acrobat PubSec 6.0, showing run-time icon and tooltip



Standard AP Dictionary

A signature's appearance is contained in the **N** (Normal) attribute of the signature annotation's **AP** dictionary. The **R** (Rollover) and **D** (Down) attributes are not used . The appearances are contained in a number of XObjects that are assembled to create layers for the appearance. These XObjects are pulled together with the **Do** page marking operator as described in "Assembling XObject Layers" on page 10.

Standard Layers

Signature appearances have five standard layers and can have a variable number of optional user-definable layers. Each layer is represented in its own XObject.

The standardized layers **n1**, **n2** and **n4** are used for run-time dynamic modification of the signature appearance, based on the validity of the signature. Adobe used these layers in Acrobat 4.0 through 5.05 but, beginning with Acrobat 6.0, no longer generates signatures that contain these layers.

The standard XObject layers are (lower numbered layers are painted first):

- **n0**—background layer, as preset when creating the signature field, for example by using the AcroForms creation tool. In all of the example appearances in "Example Signatures" on page 6, these layers are blank.
- **n1**—validity layer, for presentation of a graphic that represents the validity of the signature. This layer is used for the unknown (e.g. question mark) and valid states.. In Figure 1.1 this is the layer that contains the yellow question mark.
- **n2**—signature appearance containing information about the signature. In Figure 1.1, this is the layer that contains the line art for the handwritten signature and the text giving the name, date, reason and location of the signature. The content of this layer can be dynamically created when the signature is created, but thereafter remains static. Specifically, it remains static when the validity state is changed.
- **n3**—validity layer, for presentation of a graphic that represents the validity of the signature when the signature is invalid (e.g. X). This is above **n2** so that the X cannot be hidden by **n2**. In Figure 1.1, this layer is blank. In Figure 1.2 this layer contains the red letter X.
- **n4** - text layer, for a text presentation of the state of a signature. In Figure 1.3 this is the layer that contains the text "Signature Not Verified", and in Figure 1.4, this layer contains the text that says "Signature Valid".

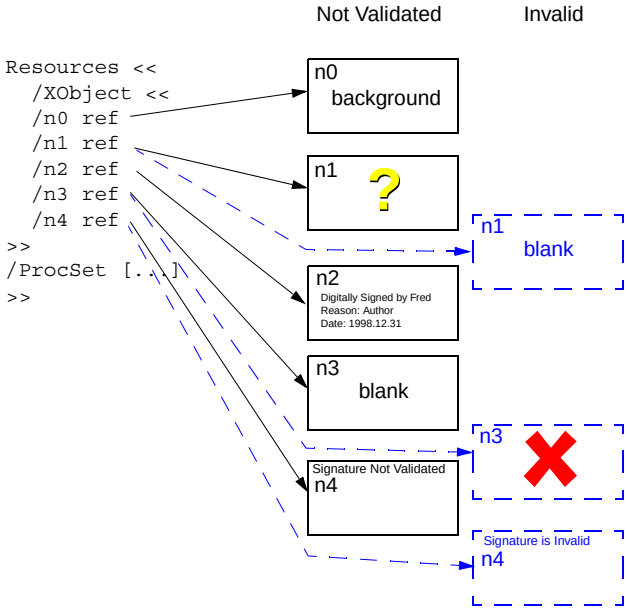
Handlers may define their own custom layers. Custom layers layers must not use the names **n0** through **n4**.

Swapping Validity States

Manipulation of siganature annotation XObjects by Adobe code will occur only if the signature appearance contains the layer **n1**, **n3** and **n4** XObjects.

A signed PDF file must always be saved to disk with validity state set to `DSSigUnknown`. In the unknown state **n1** points to the unknown logo (yellow question mark) and **n3** points to a blank XObject. When a signature is validated the references for **n1** and **n3** change to point to in-memory XObjects. If the validity state is `valid` then **n1** is set to the valid logo (usually customized by the handler) and **n3** is set to blank. If the validity state is `invalid` then **n1** references a blank XObject and **n3** points to an XObject that contains the red X.

FIGURE 1.7 Figure 6- Swapping Validity States.



When the **DSUnValidateSigProc** callback invalidates a signature, the handler must swap back **n1** and **n3** to the unknown state. The viewer thereafter calls **CosObjRevert** to force the in-memory signature annotation **AP** dictionary to match the on-disk **AP** dictionary.

You should not change the in-line stream marking operators in the second-level XObject (for example, object 18 in Figure 1.9) when swapping appearance state. You should set the matrix transformations and bounding boxes of the swapped-in XObjects accordingly.

Assembling XObject Layers

Top-level XObject

The **AP** dictionary's **N** attribute references a top level XObject. This top-level XObject is necessary to properly resize signature appearances when these appearances are referred to by more then one signature field (see object 17 in Figure 1.9 for an example). The top-level XObject performs a **Do** on our second-level XObject as follows:

```
q 1 0 0 1 0 0 cm /FRM Do Q
```

The in-line **cm** matrix may change if the signature is resized. The **Matrix** is unity (x,y translation of zero, scale factor = 1), and the **BBox** is the current bounding box of the annotation.

Adobe recommends that signature fields never refer to more then one annotation. This is recommended because the location of a signature on a page of a document can have a bearing on its legal meaning. If there is more then one location that is associated with a

signature then the meaning may loose conciseness. Regardless, it is recommended that all signatures include the top-level XObject.

Second-level XObject

The second-level XObject stream appears as follows:

```
q 1 0 0 1 0 0 cm /n0 Do Q
q 1.791 0 0 1.791 64.45 9.95 cm /n1 Do Q
q 1 0 0 1 0 0 cm /n2 Do Q
q 1.791 0 0 1.791 64.45 9.95 cm /n3 Do Q
q 1 0 0 1 0 0 cm /n4 Do Q
```

Each of **n0** through **n4** appears by name in the Resource dictionary XObject dictionary of the second-level XObject's dictionary. The **Matrix** of this XObject is unity and the **BBox** is that of the original annotation.

Beginning with Acrobat 6.0, it is recommended that layers **n1**, **n3** and **n4** no longer be used. In this situation the XObject stream will appear as follows:

```
q 1 0 0 1 0 0 cm /n0 Do Q
q 1 0 0 1 0 0 cm /n2 Do Q
```

Layer n0

This layer renders the background and border of the annotation. The attributes that specify the background and border exist in the signature field dictionary: these attributes are not set if the signature is created on-the-fly by the viewer's DigSig plug-in; however, if AcroForms was used to create a blank signature field then these values are set.

There are AcroForms HFT calls to aid in the process of obtaining these values. Refer to, for example, **`AFPDWidgetGetAreaColors`** and **`AFPDWidgetGetBorder`** to extract border and background information. There are other AcroForms HFT calls to create the **n0** XObject. These API calls are documented in *Acrobat FormsAPI Reference*.

NOTE: See "DigSigHFT Appearance APIs" on page 15 for information on the new DigSigHFT calls that can be used to create appearances.

If using a blank **n0** layer then simply call to obtain this blank object.

```
DigSigGetStdXObj ( cosDoc, DSBlankXObj ) ;
```

Layer n1

This layer is saved to disk with state set to unknown. The handler can use

```
DigSigGetStdXObj ( cosDoc, DSUnknownXObj ) ;
```

to get the standard XObject question mark (which is implemented as a Helvetica-Bold question mark with yellow fill and thin grey stroke). The handler can also specify its own custom unknown state XObject. The standard XObject is maintained by the viewer.

The in-line **cm** transformation that appears in the stream is conjured on the basis of expecting a 100 point by 100 point XObject that is to be centred in the annotation bounding box. It is therefore important, when swapping for **n1** the XObject used to represent the valid state, that this XObject fit in a 100 by 100 bounding box.

The **Matrix** and **BBox** for this XObject will be set in such a way as to bring the XObject into a bounding box that is defined by [0 0 100 100]. The standard digital signature XObjects, for example, have a **BBox** of [0,0,100,100] and unity **Matrix**.

This layer is not created by Adobe signature handlers beginning in Acrobat 6.0.

Layer n2

The handler is free to create any appearance it wishes for this layer. The layer is static: it is not changed when the validity state of the signature is changed.

PPKLite 4.0 and 4.05 came with two appearance handlers: text and picture. The text handler created a text-only appearance as per Figure 1.2. The picture handler created a combination text and graphic appearance as per Figure 1.1. The graphic used in the picture handler can be imported from any PDF file and therefore can be line art, text, images, or any combination of PDF elements.

PPKLite 5.0 has one, configurable appearance handler that replaces the functionality of the two handlers in 4.05. The handler combines text and graphics and also includes layer **n4**.

PPKEF 4.05 also supported layer **n4**.

All appearance handlers in PPKLite and PPKEF that render text honour the font type and colour defaults that were set for the signature annotation. These defaults can be obtained by calling **`AFPDFieldGetDefaultTextAppearance`** as defined in *Acrobat Forms API Reference*.

Layer n3

This layer is configured as per layer **n1**. However the unknown state for this layer should point to the **`DSBlankXObj`** XObject. Only if a signature becomes invalid will this layer be changed. In this event you should change the layer to reference either the standard XObject or a custom invalid logo.

This layer is not created by Adobe signature handlers beginning in Acrobat 6.0.

Layer n4

This is an optional layer that is used only by PPKEF 4.x and PPKLite 5.0. If you use it, you should create this layer with a **BBox** that uses absolute coordinates to define the rectangle of the appearance. In Figure 1.3 the **BBox** defines a rectangle that is the top 25% of the annotation bounding box. The **Matrix** of this XObject is whatever is required to fit the appearance into the rectangle defined by **BBox**.

This layer is intended to contain textual information about the validity of the signature. It is up to the handler to define the strings for this text, dependent on the validity state of the signature. You should use the default text attributes for the annotation when rendering the text. You can get these by using **`AFPDFieldGetDefaultTextAppearance`**.

When changing the appearance state of layer **n4** the handler should read the value of the **BBox** and create a new XObject that uses the same **BBox**. A handler can also choose to not support layer **n4**, in which case the handler should set **n4** to the DSBlankXObj XObject when the validity state is known.

This layer is not created by Adobe signature handlers beginning in Acrobat 6.0.

Signature Annotation with No Appearance

PDF signatures can be visible or invisible. In both situations there is an associated annotation. In the case of invisible signatures the annotation has a rectangle array with values [0 0 0 0]. This annotation is usually attached to the page that is being viewed when the signature is created. Despite not having an appearance, the annotation **AP** and **N** dictionaries may be present in some versions of Acrobat. If present, **N** references the **`DSBlankXObj`** (blank) XObject.

Signature Objects

Figure 1.8 shows the relationship of the signature objects in an example PDF file. The objects that contain the appearance for this example PDF file are shown in Figure 1.9:

FIGURE 1.8 Signature Objects in a PDF File

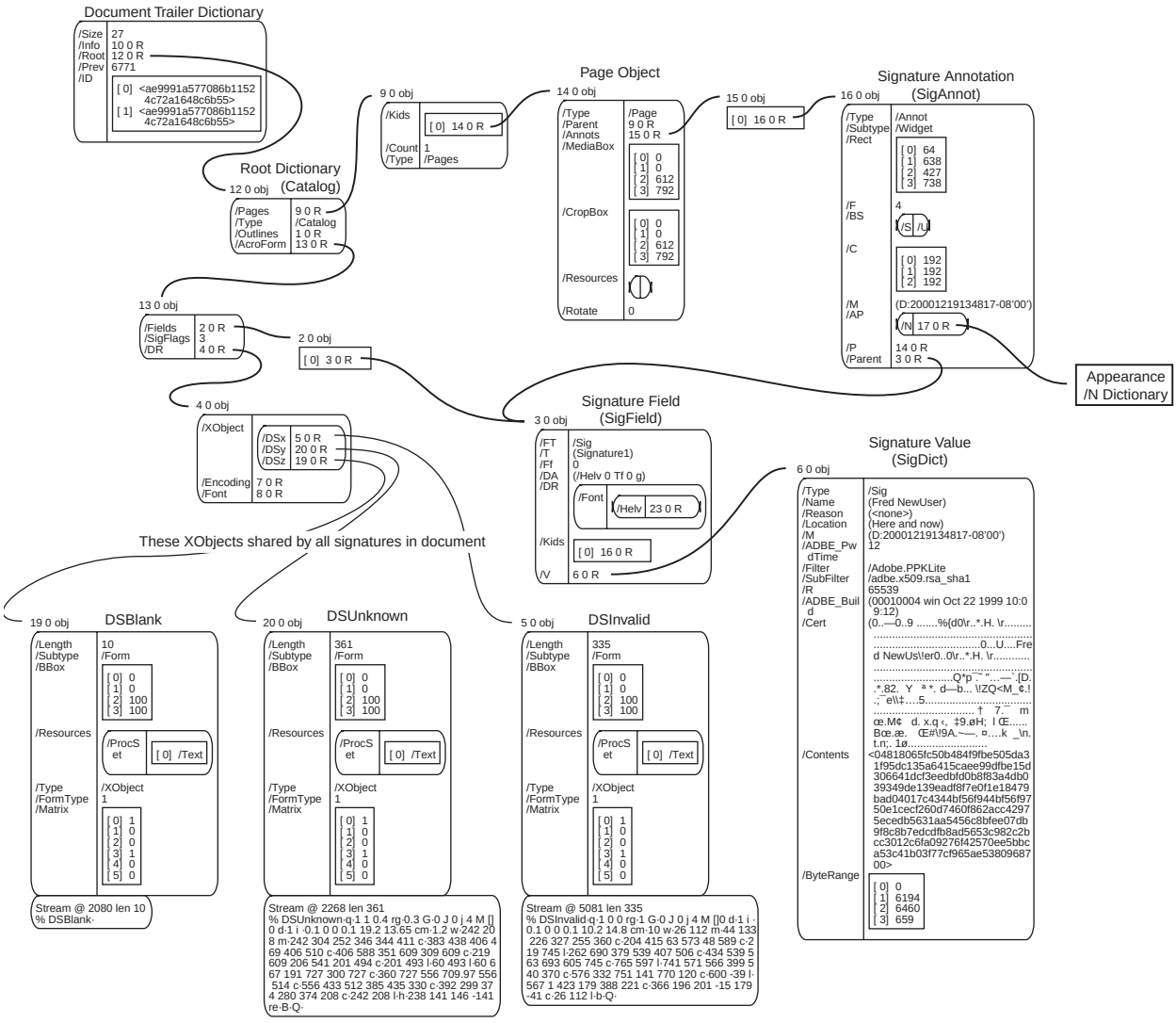
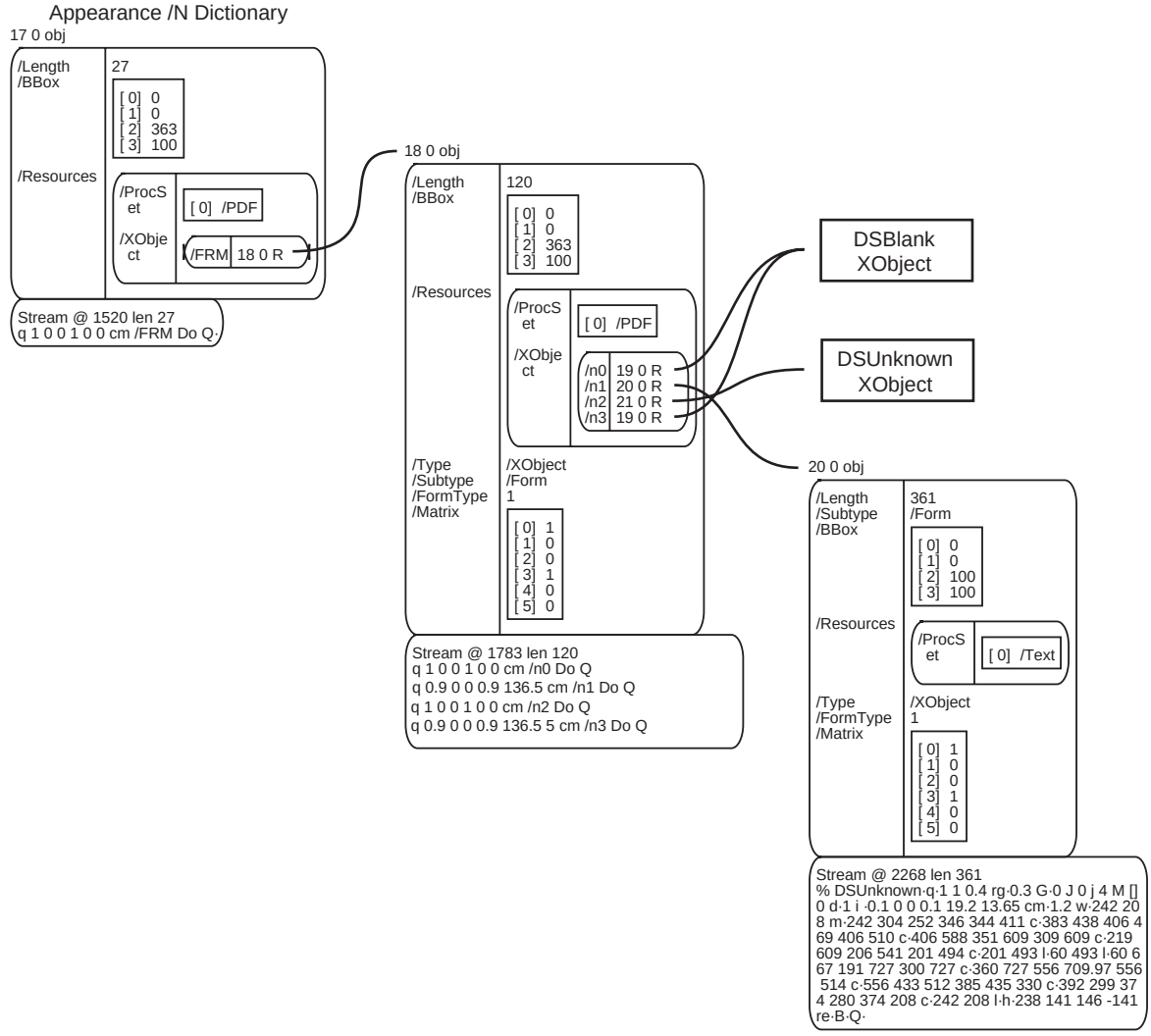


FIGURE 1.9 Signature Appearance Objects in PDF File



DigSigHFT Appearance APIs

Acrobat 5.0 introduced new API calls in the DigSigHFT that makes generation of appearances easier for third party developers. These procedures also make it easier for third party developers to create signatures that conform to the layered XObject approach that is described in this document. The procedures are declared in DigSigHFT.h and documented in the *Acrobat Digital Signature API Reference*. These new calls are:

• **`DigSigCreateStdXObj`** – Creates a standard XObject from one of the values in the enum **`DSXObjType`**. Previously only the blank, invalid and unknown XObjects could be created and, when they were created, they were added to the document (see objects 5, 19 and 20 in Figure 1.8). This new call allows a handler to get the predefined XObject for valid

and double valid signatures. In Acrobat 5.0 these are a question mark that covers a face, and a check mark.

• **`DigSigAPCreateLayeredStream`** – This actually creates the **N** dictionary and lower level XObjects given the XObjects for the various layers. The layers have defaults, so you are only required to define layer **n2**, but you will also likely want to define layer **n4**.

• **`DigSigAPXObjectFromXObjList`** – Creates a single XObject from multiple components. Use this routine to combine graphics and text components to make layer **n2**. Controls are given for placement of the graphics and text within the annotation.

• **`DigSigAPXObjectFromLogo`** – A helper routine to create an XObject from a string that contains the marking operators needed to draw an object. Adobe's signature handlers use this routine to create the Acrobat logo that is used as a signature watermark. As an example, the string that is used to create the logo is created by concatenating the strings shown in Example 1.1.

• **`DigSigAPCreateCompositeTextXObj`** – You can use this to combine various text elements into a single text XObject. PPKLite 5.0 uses this routine to combine several lines of signature validation status into a layer **`n4`** XObject. This call calls **`DigSigAPXObjectFromXObjList`** to do the actual stitching of text elements.

EXAMPLE 1.1 Example string to use with DigSigAPXObjectFromLogo

```
// Precise bbox of Acrobat logo stream stmLogoLogo
const ASFixedRect CSSEngine::stmLogoBBox =
{ 0,0,0x00630000,0x00630000 }; // 0,0,99,99
// Stream data for Acrobat Logo
const char* const CSSEngine::stmLogoData[] = {
    "% Acrobat Logo",
    "q",
    "0 J 0 j 4 M []0 d",
    "1 0 0 1 0 -1 cm",
    "1 0.85 0.85 rg",/* faded red */
    "0 i",
    "96.1 24.7 m",
    "96.4 24.7 l",
    "96.8 24.7 97 24.6 97 24.3 c",
    .
    .
    .
    .
    .
    "51.4 45.2 60.3 39.4 65.1 36.6 c",
    "53.5 34.6 40.1 31.1 28.1 25.9 c",
    "h f",
    "Q",
    NULL
};
```

Acrobat 6.0 Appearance Management

For Acrobat 6.0, the authoring of public key security handlers has been simplified by the introduction of common signature appearance code in what is called the *PubSec* software module. PubSec is a layer of code between the DigSigHFT and the PubSecHFT. Plug-in authors can allow and are encouraged to allow PubSec to take care of all signature appearance generation and maintenance, but can continue to create their own signature appearances if they so desire.

See the *Acrobat Digital Signature API Reference* for more information on the PubSec API.

Appearance File

PubSec maintains a user-configurable database of signature appearances that be used when signing a document. Users configure signature appearances using the **Edit > Preferences > Digital Signatures** menu item. The PubSec signing dialogs access this same database, allowing users to configure and choose the signature appearance that will be used when signing. The database is a PDF file called appearances.acrodata that, for Windows, is stored in the Application Data section of a user's computer. An example path is `C:\Documents and Settings\jpravetz\Application Data\Adobe\Acrobat\6.0\Security\appearances.acrodata`.

The PubSec signature appearance database is accessible via the **`DSAPFile`** interface of the PubSecHFT. The **`DSAPFile`** interface exposes enough calls to allow maintainance of the database. Maintenance includes enumeration, addition, modification and deletion calls.

`DSAPFileAcquire` – Acquire the **`DSAPFile`** object for PubSec's signature appearance database file. If the file has not already been acquired then it will be opened. PubSec acquires the database file whenever it is needed by PubSec, so handlers do not need to acquire this file unless it needs to access the file for other reasons.

`DSAPFileRelease` – This will release the file, with the potential to close and/or save the file.

`DSAPFileSave` – This will save the file if it is dirty, leaving the file open.

`DSAPFileGetCount` – Return a count of the number of configured appearance entries in the appearance database file.

`DSAPFileCanDeleteNthEntry` – Return true if this entry in the **`DSAPFile`** can be edited (that is, is not read-only). Index values that are less then zero correspond to the default appearance, which is called the StandardText appearance in the user interface.

`DSAPFileGetNewNthName` – Get a copy of the name of the Nth appearance object in the file. Use this when building a list of appearances for a user to choose from or edit.

`DSAPFileRemoveNthEntry` – Deletes the Nth appearance entry from the file.

`DSAPFileEditNthEntry` – Bring up user interface that allows the user to edit the Nth entry of the appearance file. Passing in indices larger than the number of entries in the appearance file will create a new entry in addition to bringing up the aforementioned user interface.

DSAPFileCopyNthEntry – Create a copy of the Nth entry in the appearance file and append the copy to the end of the list of signature appearances in the file. When copying default appearance entries, the resulting copy will not be considered a default appearance entry.

Annotation Appearance

Signatures that are created by PubSec continue to use the layered XObject approach that is described in this document. New signatures generated by Acrobat 6.0, however, no longer populate the n1, n3 and n4 layers of the signature appearance.

EXAMPLE 1.2 New Acrobat 6.0 signature appearances are created using only layer n0 and n2

```
q 1 0 0 1 0 0 cm /n0 Do Q
q 1 0 0 1 0 0 cm /n2 Do Q
```

PubSec continues to support validation of legacy signatures that use the n1, n3 and n4 layers. PubSec will, at run-time change these layers to show the validy state, as was done for earlier versions of Acrobat. PubSec looks for presence of these layers to determine if run-time appearance modification is to be done.

Appearance Watermark

PubSec allows customization of the watermark that is shown as part of a signature appearance. Customization can be done by a PubSec handler or by a systems administrator using the `SignatureLogo.pdf` file described below.

The watermark is incorporated as part of the **n2** layer of the signature appearance. In Figure 1.5 layer **n2** is composed of the watermark, which is the logo from the *US Department of the Interior*, plus the text "Digitally signed by ...", and the signer's name, Inky Pinky.

To customize the watermark, a systems administrator can place a file called `SignatureLogo.pdf` in the `Security` folder. If this file exists, the first page of this file is extracted and used for the watermark. If this file does not exist, PubSec handlers can provide the watermark using the PubSec API **PSGetLogo** procedure call. Adobe PubSec handlers provide the Acrobat logo as a watermark. If the **PSGetLogo** procedure is NULL, no logo watermark is added to the signature appearance.

The `Security` folder is the same folder where the `appearances.acrodata` file is stored. An example path to the appearance file is `C:\Documents and Settings\jpravetz\Application Data\Adobe\Acrobat\6.0\Security\appearances.acrodata`. For step 12 you will also need to refer to the `Stamps` folder. This will be at the same level as the `Security` folder, for example at `C:\Documents and Settings\jpravetz\Application Data\Adobe\Acrobat\6.0\Stamps`.

A watermark PDF file can be created from the line art for a company logo using Adobe Illustrator 10 and following the instructions that begin at step 1 below. A watermark can be created from an image by following the instructions at step 9 below.

To create a watermark from line art using Adobe Illustrator 10:

1. Open a file containing the company logo using Adobe Illustrator 10.0.
2. Isolate the company logo and copy it to the clipboard.
3. Create a new document and paste the logo in to this new document.
4. Select all. Note that if your logo has objects that are drawn over top of other objects, the underlying objects may be become visible through the higher layers. In this situation you may have to group the overlapping objects before proceding to the next step.
5. Open the menu **Window > Transparency** and slide the transparency slider to approximately 20%. Note that you will want the appearance to be sufficiently subdued, so a value of less then 20% may be necessary for some logos.
6. Execute the menu item **Object > Flatten Transparency**, leaving the Raster/Vector balance at 100%.
7. Save the file to a PDF file. This can be done by printing through Distiller.
8. Open the PDF file in Acrobat Pro and crop the page using the **Document > Pages > Crop** tool. Check the remove whitespace option. At this point the file may be ready to be used as the `SignatureLogo.pdf` file, and you can try to see if it is ready by going to step 10. In some instances it will not be ready and will need to be processed by the stamp tool (see step 12).

Watermarks can also be created from image files, such as bitmaps or JPEG files. To create a watermark from an image file:

9. Drag the image file (e.g. `.jpg` file) into Acrobat and create a new PDF file. Save the new PDF file. Go to step 12.

To test your watermark PDF file:

10. Copy or save your watermark PDF file as `SignatureLogo.pdf` in the Security folder.
11. Preview the watermark using the **Preference > Digital Signatures** dialog. Click the New button to create a new signature appearance. Check the logo checkbox. Your watermark should appear in the preview. If the preview is blank (grey with no image) then your watermark file is not properly prepared: you will need to go to step 12 to prepare the PDF file. Try signing a document to further verify that the watermark is working. Again, if it is not working, which will be noted by an *Internal Error* alert, then go to step 12. If your watermark appears in the signature then it is working.

Not all PDF files can be used as signature watermarks in Acrobat 6.0. In a later release we will likely fix this so that any PDF file can be used. The watermark code is expecting a specific structure to the first page contents. If the structure is not correct then the file structure will need to be prepared. This can be done using the stamp commenting tool.

To further prepare your watermark file using the commenting stamp tool do the following:

12. Note the contents of the *Stamps* folder.
13. Execute the menu command **Tools > Commenting > Stamp Tool > Create Custom Stamp**. Select your watermark file and import it as a new stamp in a new category.

14. Note the new stamp file that has been added to the Stamps folder. Copy this file to the Security folder and rename it SignatureLogo.pdf. Go to step 10 to test your watermark file.

Adobe does not provide support for logo watermark preparation or use. If you have trouble with particular aspects of this procedure then consult a graphics professional.