



In This Issue

Volume 5, Number 5

p . 1

Everything You Wanted to Know About Adobe® Acrobat® Annotation Handlers But Were Afraid to Ask

p . 2

How to Reach Us

p . 6

PostScript® Language Technologies

Everything You Wanted to Know About Adobe Acrobat Annotation Handlers But Were Afraid to Ask

Adobe Acrobat Exchange™ has a powerful plug-in API that allows you to modify its user interface or expand the range of data types used in PDF files. One way you can add value is by writing a plug-in that adds a new annotation subtype. An annotation is something that floats above the page contents and is not part of the PDF page itself. Examples of the built-in annotations types are links and text notes. A few examples of new annotations are:

- a markup on top of a PDF page. This could be a “pen” that the user draws with on top of the PDF file, and the markings are stored in an annotation.
- a new special type of note. This could be a custom text note that allows the user to pick different fonts and colors for the text in the note.
- a paperclip on a particular page. This could be a small icon at the top of a page that allows the user to cycle through the “paperclipped” pages

This article explores the various parts of the annotation handler (the C code that creates, modifies, and displays the annotation) and demonstrates how to register an annotation handler with the Acrobat viewer. If there are any unfamiliar Acrobat API concepts in this article, please look at the Acrobat column in the *ADA News* volume 5, number 4. They describe the basics of setting up a plug-in and filling in Acrobat callbacks.

Description of Annotation Handler and its Methods

The `AVAnnotHandlerRec` structure must be filled in to register a new annotation. This structure is defined in the `AVExpt.h` header file in the `PLUGINS:HEADERS` directory on the Acrobat Plug-ins SDK. See the document `VWRPIREF.PDF` in the `TECHDOCS` directory for a full description of the `AVAnnotHandler`. The annotation handler record is made up of the following data and callbacks:

- `os_size_t (data)`—specifies the size of the annotation handler record. Newer Acrobat viewers will add additional methods on to this structure, so the size field is required to specify the version of the viewer that your plug-in will work with.
- `Uns32 (data)`—specifies a collection of flags affecting the annotation’s behavior. The only flag currently defined is `ANNOT_CLIP_TEXT_SELECTION`. If this flag is set, text selection in the main document never selects text within the annotation. Another flag affects whether an annotation is visible if the annotation handler is not present; you can get and set this flag using `PDAnnotGet/SetFlags()`.
- `AVAnnotHandlerGetTypeProc (callback)`—returns an `ASAtom` indicating the annotation type for which the handler is responsible. It corresponds to the annotation’s subtype key in the PDF file.
- `AVAnnotHandlerDoClickProc (callback)`—handles mouse clicks within the annotation region.

continued on page 2

How To Reach Us

DEVELOPER INFORMATION ON THE

WORLD WIDE WEB:

www.adobe.com

See the Support & Services section and point to Developer Relations.

DEVELOPERS ASSOCIATION HOTLINE:

U.S. and Canada:

(415) 961-4111

M-F, 8 a.m.-5 p.m., PDT.

If all engineers are unavailable, please leave a detailed message with your developer number, name, and telephone number, and we will get back to you within 1 work day.

Europe:

+31-20-6511-355

FAX:

U.S. and Canada:

(415) 967-9231

Attention:

Adobe Developers Association

Europe:

+31-20-6511-313

Attention:

Adobe Developers Association

EMAIL:

U.S.

ada@mv.us.adobe.com

Europe:

eurodevsup@adobe.com

MAIL:

U.S. and Canada:

Adobe Developers Association

Adobe Systems Incorporated

1585 Charleston Road

P.O. Box 7900

Mt. View, CA 94039-7900

Europe:

Adobe Developers Association

Europlaza

Hoogoorddreef 54a

1101 BE Amsterdam Z.O.

The Netherlands

Send all inquiries, letters and address changes to the appropriate address above.

Everything You Wanted to Know

- `AVAnnotHandlerDoKeyDownProc` (callback)—handles key presses in the annotation region.
- `AVAnnotHandlerDrawProc` (callback)—draws the annotation.
- `AVAnnotHandlerAdjustCursorProc` (callback)—controls the cursor shape when the cursor is within the annotation.
- `AVAnnotHandlerDoPropertiesProc` (callback)—displays whatever user interface is required to allow a user to change an annotation's properties. This method is called when the user selects the "Properties..." item from the "Edit" menu while an annotation of this type is selected.
- `AVAnnotHandlerNotifyAnnotAddedToSelectionProc` (callback)—notifies the annotation handler when an annotation is added to the selection and should highlight the annotation.
- `AVAnnotHandlerNotifyAnnotRemovedFromSelectionProc` (callback)—notifies annotation handler when an annotation is removed from the selection and should unhighlight the annotation.
- `AVAnnotHandlerGetLayerProc` (callback)—returns the annotation's layer. Note that the layer does not need to be a constant. Annotations that have a high layer value appear on top of annotations that have a lower value. For example, the text note has two different modes: open and closed. The open mode has a higher layer value than the closed mode, therefore, an open text note will appear on top of a closed text note.
- `AVAnnotHandlerGetAnnotViewBBoxProc` (callback)—returns the rectangle enclosing the annotation on the screen.
- `AVAnnotHandlerPtInAnnotViewBBoxProc` (callback)—determines whether or not a point is within an annotation. The annotation handler is free to determine what it means for the point to be "in" the annotation.

Annotations can be rectangular or non-rectangular shapes. The difference between the methods `GetAnnotViewBBoxProc()` and `PtInAnnotViewProc()` is that `GetAnnotViewBBoxProc()` returns the rectangle containing the annotation, while `PtInAnnotViewBBoxProc()` determines if the cursor is actually within the annotation. As soon as the viewer detects the cursor entering the annotation's rectangle, it calls the `PtInAnnotViewBBoxProc` to allow a developer the chance to determine if the cursor is within the specified annotation region. As an example, let us say that we have a circular annotation. The `GetAnnotViewBBox` method returns the rectangle that encloses the circle, while the `PtInAnnotViewBBox` method determines if the cursor is within the circle.

There are two additional `AVAnnotHandler` methods, `AVAnnotHandlerNewProc` and `AVAnnotHandlerNotifyDestroyProc`, that are unused by the Acrobat 2.1 viewer. Set the corresponding entries in the structure to `NULL`.

Everything You Wanted to Know

The only method that you must provide in an `AVAnnotHandlerRec` structure is `GetTypeProc`. If you do not define this method, the annotation handler is not registered. However, to have a useful annotation, you should at least define a `DrawProc` to draw your annotation, an `AdjustCursorProc` to set the cursor shape when it is over the annotation, a `GetAnnotViewBBoxProc` to define the outer bounding box of the annotation, and a `PtInAnnotViewBBoxProc` to define the actual annotation region.

You should draw your annotation on the PDF page in the `DrawProc` method. You can either use Acrobat API methods to draw your annotations, which works across multiple platforms, or uses platform-specific drawing calls, which result in a solution that is platform dependent. You have a choice of either resizing your annotation as the user changes the view percentage (like the Acrobat viewer's link annotation) or maintaining a constant size at all times (like the Acrobat viewer's note annotation). Acrobat versions 2.1 and earlier do not support printing of annotations when pages are printed, but future versions may support this.

Creating and Destroying Annotations

In order to add an annotation to a page, you must either create an Acrobat Tool (which allows a user to use the mouse to create an annotation) or implement a batch process to automatically create annotations. Using a Tool to add an annotation to a page involves determining and storing the bounding box of the new annotation and storing any private data you require. Private data is any information you need to represent your annotation; this could be the radius of a circle, a text string, or a series of x-y coordinates to designate markup on a page. The use of Tools in the Acrobat viewer will be discussed in greater detail in a future article.

There are several ways to batch create annotations. One method is to create a menu item that can be called from an external application to create annotations or to create a plug-in that makes annotations. Another technique is to implement an Interapplication Communication method to automatically make the annotations. A third and final way is to use **pdfmark** in a PostScript language file in order to have the Acrobat Distiller® application create your annotations.

Removing an annotation from a page is simple. The viewer will automatically delete the entire annotation object in the PDF file, along with any data that you store in it.

Sample Annotation Handler

The following sample code demonstrates how to initialize an annotation handler and set up a few of its important methods. The sample annotation draws a rectangle bounding the annotation region using the Acrobat API method `AVPageViewDrawRectOutline()`, which is used to draw an outline of a rectangle. Note that this code only registers the annotation; it does not demonstrate how to add the annotation to a PDF file (see the previous paragraph for ideas on adding annotations).

Everything You Wanted to Know

`sampleAH` is the annotation handler that we will fill in. `k_SampleAtom` is the name of the annotation, stored as an `ASAtom`. It is initialized in `MyInitProc`.

```
AVAnnotHandlerRec sampleAH;
ASAtom k_SampleAtom;
```

The `GetTypeProc()` is used to return the name of the annotation handler you want to register.

```
static ACCB1 ASAtom ACCB2 SampleGetType (AVAnnotHandler handler)
{
    return k_SampleAtom;
}
```

The `ViewBBoxProc()` fills in the bounding box of the annotation. `PDAnnotGetRect()` returns the rectangle enclosing the annotation as it is stored in the PDF file. This must be converted from a `FixedRect` to an `AVRect` (two different coordinate systems used in the Acrobat viewer), which is what `AVPageViewRectToDevice()` does.

```
static ACCB1 void ACCB2 SampleGetViewBBox (AVAnnotHandler annotHandler,
AVPageView pageView, PDAnnot annot, AVRect *bbox)
{
    FixedRect pdRect;

    PDAnnotGetRect(annot, &pdRect);
    AVPageViewRectToDevice(pageView, &pdRect, bbox);
}
```

The `DrawProc()` draws the annotation—in this case, it draws a rectangle around the annotation region. We know that the method `SampleGetViewBBox()` gives us the rectangle representing the annotation. We can then use the Acrobat API method `AVPageViewDrawRectOutline()` to draw the rectangle.

```
static ACCB1 void ACCB2 SampleDraw (AVAnnotHandler handler,
PDAnnot annot, AVPageView pageView)
{
    AVRect stampRect;

    SampleGetViewBBox (handler, pageView, annot, &stampRect);
    AVPageViewDrawRectOutline(pageView, &stampRect, 5, NULL, 0);
}
```

The `MyInitProc()` is called by the plug-in during plug-in initialization. We initialize `k_SampleAtom` to the `ASAtom` equivalent to the string “MySampleAnnot”, and initialize the `AVAnnotHandler` structure with zeroes. The size field must be set to the size of an `AVAnnotHandlerRec` for future compatibility. Each of the annotation handler’s methods that are being used are then set using

C o l o p h o n

This newsletter was produced entirely with Adobe PostScript® software on Macintosh® and IBM® PC compatible computers. Typefaces used are from the Minion® and Myriad® families from the Adobe Type Library.

Managing Editor:
Jennifer Cohan

Technical Editor:
Nicole Frees

Art Director/Designer:
Karla Wong

Contributors:
Dave Hackel, George Ishii

Adobe, the Adobe logo, Acrobat, Acrobat Exchange, Distiller, Minion, Myriad, PostScript, and the PostScript logo are trademarks of Adobe Systems Incorporated. Macintosh is a trademark of Apple Computer, Inc. registered in the U.S. and other countries. IBM is a registered trademark of International Business Machines Corporation. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company. Times is a registered trademark of Linotype-Hell AG and/or its subsidiaries. All other brand and product names are trademarks or registered trademarks of their respective holders.

©1996 Adobe Systems Incorporated.
All rights reserved.

Part Number CS0278 7/96



Adobe PostScript

Everything You Wanted to Know

the Acrobat plug-in method `ASCallbackCreateProto()`, which is used for type checking. Lastly, we register the annotation handler using `AVAppRegisterAnnotHandler()`. This checks to make sure there is a `GetTypeProc` for the annotation, and overrides any annotation handlers that were previously registered with that type.


```
static ACCB1 boolean ACCB2 MyInitProc (void)
{
    memset(&sampleAH, 0, sizeof(AVAnnotHandlerRec));

    k_SampleAtom = ASAtomFromString("MySampleAnnot");

    sampleAH.size = sizeof(AVAnnotHandlerRec);
    sampleAH.flags = 0; /* set flags to NULL */

    sampleAH.GetType = ASCallbackCreateProto(AVAnnotHandlerGetTypeProc,
        &SampleGetType);
    sampleAH.Draw = ASCallbackCreateProto(AVAnnotHandlerDrawProc,
        &SampleDraw);
    sampleAH.GetAnnotViewBBox =
        ASCallbackCreateProto(AVAnnotHandlerGetAnnotViewBBoxProc,
            &SampleGetViewBBox);

    AVAppRegisterAnnotHandler(&sampleAH);
    return true;
}
```

For more information on creating annotations in your plug-ins, see the source code sample Stamper contained in the Acrobat Plug-Ins SDK. Also, the documents *VWRPIREF.PDF* and *VWRPIOVR.PDF*, located in the TECHDOCS folder on the Acrobat Plug-Ins SDK, contain information on registering and using annotation handlers and their associated methods. For further information on developing plug-ins for the Acrobat viewer, please contact the Adobe Developers Association and request an Acrobat Developer Information Kit, or look up the information on our website at <http://www.adobe.com>. Look in the Support & Services section and point to Developer Relations. 

PostScript Language Technologies

In this month's column, we respond to a question asked by a member of our PostScript Language Technologies program.

Q I would like certain characters in my Type 1 font to be vertically shifted. I am aware of utilities that change the font, but I would prefer to do this dynamically in my PostScript language code. My program runs on a UNIX® platform and I am not concerned about the display of the characters on the screen.

A You can modify the metrics of any character in a Type 1 font. This is done by adding a Metrics dictionary to the font dictionary as documented in Section 5.6.2 of the *PostScript Language Reference Manual, Second Edition*. The optional Metrics dictionary contains metric information for one or more characters. Adding a Metrics dictionary to a Type 1 font overrides the metric information for the specified characters. For a review of Type 1 character metrics, see section 5.4 of the *PostScript Language Reference Manual, Second Edition*.

The following steps can be used to modify a font's character metrics:

1. Make a copy of the original font minus the /FID and /UniqueID entries.
2. Make a copy of the CharStrings dictionary, using the **copy** operator, not the **dup** operator.
3. Create the Metrics dictionary to include metrics for the characters which you would like to change. The Metrics dictionary consists of one or more key-value pairs. Keys are the names of individual characters referenced by their standard roman character set names as listed in section E.5 of the *PostScript Language Reference Manual, Second Edition*. Values must be either a single number, an array of two numbers, or an array of four numbers, where:
 - A single number indicates a new horizontal width.
 - An array of two numbers indicates the horizontal components of a new left sidebearing and new width.

- An array of four numbers indicates the horizontal and vertical components of the left sidebearing followed by the horizontal and vertical components of the width.
 - All of these are specified in character space units, which are normally equal to 1/1000 of an em.
4. Define the font with a new name using the **definefont** operator.

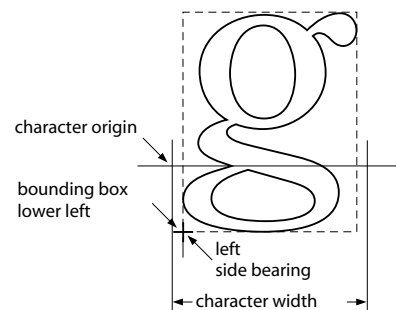


Figure 1 Character Metrics Example

Example 1 on page 7 follows the steps indicated above to copy the Times-Roman font, and create a new font, MyNew-Times. In this new font, characters 'b', 'g' and '7' are vertically shifted. The Metrics dictionary contains three entries. The keys 'b', 'g', and 'seven', are each associated with an array of four numbers, which are determined as follows:

- The first number in the array (the horizontal component of the left sidebearing) should be the same as the horizontal component of the lower left corner of the character bounding box (see figure 1). This number can be found in the font's AFM file as the first value of the character's bounding box (the four numbers after the 'B' key). In Times-Roman, this value for the 'g' character is 28, as shown in figure 2.

from AFM file: C 103 ; WX 500 ; N g ; B 28 -218 470 460 ;

Metrics dictionary entry: /g [28 -250 500 0]

negative vertical character shift of 1/4 of an em

Figure 2 Mapping AFM values to Metrics dictionary entries

- The second number in the array (the vertical component of the left sidebearing) is equal to the amount of positive or negative vertical shift. The example specifies a negative vertical shift of 250 (¼ of an em) for the ‘g’ character.
- The horizontal component of the character width, which is the third number in the array, should be the same as the horizontal character width; this number can be found in the font’s AFM file as the value after the ‘WX’ key. This value is 500 for the Times-Roman ‘g’ character, as shown in figure 2.
- The fourth value in the array is the vertical component of the character width which is normally 0 for Roman fonts.

The Example 1 code prints a line of text in Times-Roman and in MyNew-Times as shown in figure 3.

abcdefg 678

a^bcdef_g 678

Figure 3 Output from Example 1

Example 1

```

%!
% makes a copy of the font dictionary without /FID and
/UniqueID entries
% including a copy of the CharStrings dictionary

/Times-Roman findfont dup length dict begin {
    1 index /FID eq 2 index /UniqueID eq or {pop pop}
    {def} ifelse
} forall

CharStrings dup length dict copy
/CharStrings exch def

% create a /Metrics dictionary with 3 entries
3 dict begin
    /b [3 250 500 0] def
    /g [28 -250 500 0] def
    /seven [20 -150 500 0] def
currentdict end
/Metrics exch def
currentdict end
/MyNew-Times exch definefont

% print text for comparison

/Times-Roman findfont 12 scalefont setfont 100 100 moveto
(abcdefg 678) show

/MyNew-Times findfont 12 scalefont setfont 100 75 moveto
(abcdefg 678) show

showpage

```

§