In This Issue

(Click on an article link to read it,
click on the ADA News header to return to this page)

Adobe

How To Reach Us

Developer Information on the World Wide Web, *http://www.adobe.com*

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

Developers Association Hotline:

U.S. and Canada:
[408] 536-9000
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 one work day.

Europe:
+31-20-6511-355

Fax:
U.S. and Canada:
[408] 536-6883
Attention:
Adobe Developers Association

Europe:
+31-20-6511-313
Attention:
Adobe Developers Association

EMAIL:
U.S.
ada@adobe.com

Europe:
eurodevsup@adobe.com

Mail:
U.S. and Canada:
Adobe Developers Association
Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110-2704

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.

Feature Article

Introduction to the Adobe After Effects 3.1 SDK

This is the first *ADA News* article dedicated to Adobe After Effects.™ In this article we'll give you an introduction to the SDK as well as a brief introduction to the program itself. Since the After Effects product will be new to many readers, we hope to give you enough information to become interested in developing plug-ins for After Effects, and if you are already an Adobe Photoshop® plug-in developer, making your Photoshop plug-ins After Effects savvy. In future editions of *ADA News* we'll present more details on the After Effects API and its SDK.

Introduction to the After Effects Program

Adobe After Effects is the most powerful desktop application program available for performing compositing, 2D animation, and special effects on the Macintosh® or Windows® platforms. As a special effects package it has become a standard tool in digital post-production for film, broadcast video, and multimedia. It combines high-resolution compositing controls for multilayer compositions, sophisticated motion control for high-quality, 2D animation, and advanced special effects capabilities for time-varying special effects. With its professional-level feature set, Adobe After Effects has no direct competition on either the Macintosh or Windows platform. This is further reinforced when the special effects and compositing tools of Adobe After Effects are combined with Adobe Premiere,® which provides nonlinear digital video and audio editing capabilities. When

Introduction to the Adobe After Effects 3.1 SDK

Adobe After Effects and Adobe Premiere are used in conjunction with Adobe Photoshop and Adobe Illustrator,® Adobe's image manipulation and illustration software, the unique combination forms the core of a professional, integrated post-production suite capable of creating and delivering every level of target content, from QuickTime® movies on CD-ROM to D-1 broadcast-quality digital video and film resolution for motion picture production effects.

Introduction to the SDK

The Adobe After Effects application supports plug-in modules for applying special effects to images and for performing file input and output. Effects modules can appear in the Effect menu or in the render queue of the program, and the controllers (sliders, pop-ups, etc.) that configure the effect appear in an effect floater. File I/O modules appear in the File menu.

The interface for Adobe After Effects plug-ins is somewhat similar to the one used in Adobe Photoshop; authors of Photoshop filters should be able to get up to speed quickly with After Effects plug-ins. However, just as Photoshop plug-ins were completely different in detail from the Silicon Beach plug-ins that preceded them, so too, After Effects plug-ins are completely different from Photoshop's. After Effects adds support for time-varying parameters, application integrated plug-in parameter control, and a library of graphical utility callbacks, and eliminates the virtual memory complications and varying image data representations.

Introduction to the Adobe After Effects 3.1 SDK

The After Effects SDK assumes you are proficient in C language programming and tools. The source code files in the toolkit are written for the Metrowerks CodeWarrior software development environment. Before undertaking plug-in development, you should have a working knowledge of the After Effects program and understand how plug–in modules work from a user's viewpoint.

After Effects Pipeline

The diagram on page 5 depicts the After Effects pipeline and shows where plug-ins fit into the picture.

Key to the Following Diagram

Cache—The item cache buffer (at the top of the diagram) holds a source item after retrieval. This helps to avoid rerendering compositions, costly 3-2 pulldown, and frame blending. A post filter cache (toward the bottom) avoids rerendering effects and speeds compositing.

Mask—Bézier shapes with feathering are matted with an existing alpha channel.

Geometry—Provides affine transformations (scaling, rotation, etc.) with motion blur including resampling and subpixel positioning.

Blend—Combines layers into a composition, this includes compositing, Photoshop transfer modes, and track mattes.

Introduction to the Adobe After Effects 3.1 SDK

eFKT—This is an After Effects effects filter for manipulating pixel images. This is a plug-in which you can write. There can be an unlimited number of these, the writing of which is the primary topic of the SDK documentation.

8BFM—This is a Photoshop filter, which may or may not contain an ANIM resource. Photoshop filters with an ANIM resource can be varied over time. Please refer to the *Adobe Photoshop Plug-In Software Development Toolkit* for information on writing Photoshop filters. The companion document *Cross-Application Plug-in Development Resource Guide* contains information on the ANIM resource.

Layer Parameters.

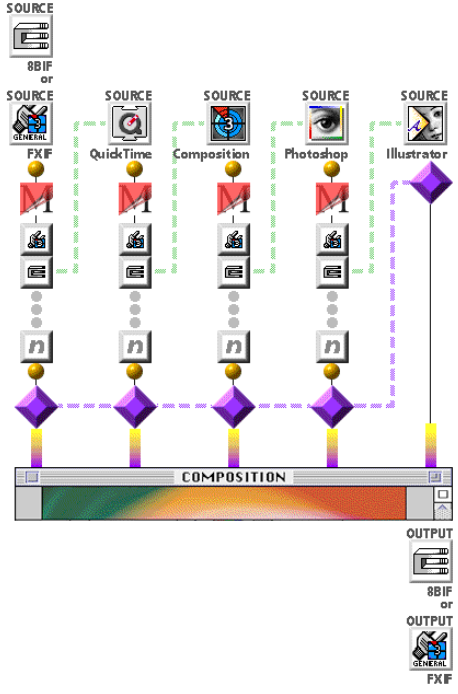
Track Matte.

8BIF—This is a standard Photoshop I/O plug-in.

FXIF—This is an After Effects I/O plug-in. This is essentially an 8BIF plug-in which has been extended. These are also described by the SDK documentation.

Introduction to the Adobe After Effects 3.1 SDK

The After Effects Pipeline



How a Plug-in is Called

The key to understanding the Adobe After Effects plug-in specification lies in recognizing that After Effects does not immediately process image data when the user chooses an effect. Instead, After Effects adds the effect to a model of what the user wants done. Only when the output of the effect is needed, to make a movie or to update the screen, does After Effects attempt to apply the effect.

If the user applies the same effect in multiple places throughout the project, After Effects creates multiple "instances" of the effect in the model. Each instance is responsible for a sequence of single frames over time. All instances of the effect can share global data. Each instance can share data between all frames in its sequence. Ultimately, the effect is invoked to render a single frame of data.

Introduction to the Adobe After Effects 3.1 SDK

To invoke an effect, Adobe After Effects looks at the plug-in's PiPL resource (more on this later) to find the entry point of the code (68K or PPC). Every plug-in has a single entry point with selectors indicating the desired operation.

Here is the prototype for the entry point of a plug-in:

```
PF_Err main (
        PF_Cmd cmd,
        PF_InData     *in_data,
        PF_OutData    *out_data,
        PF_ParamList  params,
        PF_LayerDef   *output,
        void          *extra );
```

The `cmd` parameter is a selector for the requested plug-in function. The various selectors are summarized below.

The `in_data` parameter points to a parameter block of information that the plug-in may wish to use. In this parameter block (among other things) are function pointers that provide a variety of interface and image manipulation services to the plug-in.

The `out_data` parameter points to a parameter block of output values that the plug-in can set to communicate various things back to the application.

The `params` parameter points to an array of structures that describe the values of the plug-in's parameters at the time of the current invocation. This first of the parameters (`params[0]`) describes the input image upon which the effect is to act. These `params` settings will only be valid for

certain function selectors and are set by the application user to specify the effects of the plug-in.

The `output` parameter points to the output image—the effect sets this image to the altered contents of the input image. Again, this parameter will only be valid for certain function selectors.

The `extra` parameter was new in 3.0 and is used by plug-ins incorporating a custom user interface.

The return value of the routine is a 32-bit integer value. It can either be a sign-extended standard Macintosh OS error code or one of a variety of values defined in the effects header file.

Command Selectors

The plug-in will typically be invoked many times in a given run of After Effects, for many different reasons. The `cmd` parameter indicates what After Effects wants from the plug-in during any given invocation. There are three scopes of invocation, the first of which is the global scope. Global commands manipulate data that will be shared everywhere that the effect is applied in a project. Next is the sequence scope. Sequence commands work with data that will be shared between every frame in a single sequence to which the effect has been applied. The difference between global commands and sequence commands is that global commands have no information about the size of the image to which they are applied, whereas sequence commands know they are being applied to a particular size and duration movie. Finally there are frame commands. These are responsible for actually producing a single rendered frame with the effect applied.

Introduction to the Adobe After Effects 3.1 SDK

The Adobe After Effects command selectors are:

Global Commands

PF_Cmd_ABOUT
PF_Cmd_GLOBAL_SETUP
PF_Cmd_GLOBAL_SETDOWN
PF_Cmd_PARAMS_SETUP

Sequence Commands

PF_Cmd_SEQUENCE_SETUP
PF_Cmd_SEQUENCE_RESETUP
PF_Cmd_SEQUENCE_FLATTEN
PF_Cmd_SEQUENCE_SETDOWN

Frame Commands

PF_Cmd_DO_DIALOG
PF_Cmd_FRAME_SETUP
PF_Cmd_RENDER
PF_Cmd_FRAME_SETDOWN
PF_Cmd_PARAMS_UPDATE

User Interface Commands

PF_Cmd_EVENT

Of these commands, PF_Cmd_ABOUT, PF_Cmd_GLOBAL_SETUP, PF_Cmd_PARAMS_SETUP, and PF_Cmd_RENDER are required and must be handled in every After Effects plug-in.

In general, on program startup, effect modules will receive the GLOBAL_SETUP and the PARAMS_SETUP selector. Each time the user chooses an effect to apply to a layer (i.e. adds an effect to the description), the effect will receive the SEQUENCE_SETUP selector. To render a frame, After Effects sends FRAME_SETUP, then RENDER, then FRAME_SETDOWN. The SEQUENCE_SETDOWN selector is sent when quitting or when the user de-applies an effect. SEQUENCE_RESETUP is potentially sent when an After Effects project is loaded in from

disk or when a layer is significantly reconfigured. The FLATTEN selector may be sent when the After Effects project is written out to disk. ABOUT is sent when the user chooses About… from the Effect menu.

Callbacks

Adobe After Effects provides several sets of callbacks: a set for user interaction, a set for ANSI C style string manipulation, a set for math intrinsics, a set of graphics utilities, a set for color space conversions, and a set for custom user interface events.

Use of callbacks can allow a plug-in to be kept compact. The callbacks themselves are efficient and have been well tested by Adobe. In addition, using the callbacks will automatically allow a plug-in to take advantage of multi-processor hardware (if available) which After Effects 3.1 now supports.

User Interaction Related

Every effect will call some of the interaction callbacks. These callbacks provide operations for adding, checking out, and checking in parameters, and for registering a custom user interface, aborting, and presenting a progress bar.

Graphics Utilities

The graphics utility callbacks perform a large number of raster operations including sampling, blend, convolve, a gaussian kernel, iteration functions, and transformations.

Plug-ins do not have to use the graphics utility callbacks, but there are advantages in doing so. First, it is frequently easier to use these callbacks rather than re-implementing the wheel. The callbacks will, in general, be relatively fast and will work correctly.

Introduction to the Adobe After Effects 3.1 SDK

Second, on machines that have a DSP or other hardware accelerator installed, some of these callbacks may be accelerated automatically, transparent to the plug-in. Third, it will make it easier to port your plug-in code to another platform.

Intrinsics

Along with the variety of graphics utilities, After Effects also provides a block of ANSI standard routines so plug-ins don't need to include the ANSI library to use standard functions. These include a host of trigonometric and string manipulation routines.

Colorspace Conversions

Colorspace conversion callbacks provide functions for converting between various color spaces such as RGB (alpha, red, green, blue), HLS (hue, lightness, saturation), and YIQ (luminance, inphase chrominance, quadrature chrominance).

Custom User Interface

To provide a custom user interface for plug-ins, After Effects 2.0 and 3.x support a modal dialog via an "options" button mechanism. After Effects 3.x also added a new facility to the plug-in API for adding non-modal custom user interface elements in the effects, composition, and layer windows.

A plug-in can inform the After Effects program what kind of UI events it wishes to receive. These events allow the plug-in to respond to click, draw, and keydown events which can occur in the comp, layer, effect, and preview windows. The plug-in can store state information associated with a given context (or window) which After Effects stores between events.

Introduction to the Adobe After Effects 3.1 SDK

The event also identifies any particular controls being modified and returns coordinate information so the plug-in can draw back into the effect, context, comp, and info windows.

There is also a set of callbacks provided for custom UI plug-ins, which perform coordinate mapping functions and drawing functions.

To see an example of a plug-in in action which uses a custom UI, try playing with the Bulge filter. This is included with the After Effects Production Bundle.

Resources

All After Effects 3.x plug-ins must now include a Plug-in Property List (PiPL) resource to provide configuration information to the host. This concept was taken from Photoshop which originated the PiPL resource. There are two major PiPL aspects of which to be aware.

- All After Effects plug-ins must include a PiPL resource which identifies the plug-in and provides flags and other static properties that control the operation of the plug-in.
- Photoshop filters can be made animatable in After Effects by adding an ANIM atom to their PiPL resource.

For detailed information on both of these topics, refer to the document entitled *Adobe Graphic Application Products, Cross-Application Plug-in Development Resource Guide*. Chapters 3 and 4 of this document cover After Effects specific information. This guide is a companion document which is included with both the Photoshop and After Effects SDKs.

Introduction to the Adobe After Effects 3.1 SDK

Making Photoshop Filters After Effects Savvy

As mentioned earlier, After Effects can accept Adobe Photoshop filters. With the addition of an ANIM atom to the Photoshop plug-in's PiPL resource, After Effects can control the filter over time. Generally this doesn't require any changes to the source code of the filter itself, only modification of its PiPL resource. We urge all Photoshop plug-in developers to make their filters animatable in After Effects by adding the ANIM atom. By doing this you can market your plug-ins to After Effects users and you don't even need to look at the After Effects SDK!

You can see this demonstrated by noticing that the After Effects program includes several Photoshop filters (Extrude, Lens Flare, Pinch, Radial Blur, Ripple, Spherize, Tiles, Twirl, Wave, ZigZag) which are animatable. These are all standard Photoshop filters which have simply had their PiPL resources modified by the After Effects engineering team.

PiPLs are complicated resources which are difficult to impossible to edit using *ResEdit.*™ The After Effects SDK includes a *Resorcerer* (a resource editing program from Mathemaesthetics, Inc.) template for graphically editing PiPLs and ANIMs.

Current SDK Status

The Adobe After Effects 3.1 SDK is the current shipping version of the After Effects SDK at the time this article is published. Like all the Adobe SDKs for graphics and publishing products, it is included on the *Adobe Graphics and Publishing SDK* CD-ROM, and is posted on the Adobe Web site (*www.adobe.com* under Support and Services, Developer Relations, SDK).

Introduction to the Adobe After Effects 3.1 SDK

While a Windows 95 and Windows/NT™ version of After Effects have been announced, the shipping SDK presently supports only the Macintosh. In looking through the SDK you'll notice it is quite platform independent, so porting plug-ins for the Macintosh version of the After Effects software to the upcoming Windows version should be a relatively painless task.

The After Effects plug-in API underwent substantial revision for release 3.0 and 3.1 of the product. There are no API differences between the 3.0 and 3.1 releases, however the 3.x API is not backwards compatible to 2.0, so older plug-ins must be modified to work with After Effects 3.x. Fortunately, we think you'll find this a straight forward conversion. §

DEVELOPING WITH

Adobe Photoshop

The Adobe After Effects Application hosting Adobe Photoshop Plug-ins

A previous column published in Volume 5, Number 3, detailed some strategies for cross-application plug-in development using the Adobe Photoshop Application Plug-in Interface (API) specification. This column will begin to focus on how other Adobe applications support the Photoshop API, and important caveats when targeting those applications. Specifically, we'll detail how the Adobe After Effects application hosts Adobe Photoshop Plug-ins.

All this information is available in the Adobe Photoshop Software Development Kit (SDK), and the Adobe After Effects SDK.

The information is available from Adobe's web site: *www.adobe.com/supportservice/devrelations/sdks.html*

Cross-application host support and callback suites

Not all applications are completely-savvy Adobe Photoshop plug-in hosts. In some cases, some applications support a much earlier version of the API. Because some callbacks didn't exist until later versions, it is important, while developing your plug-ins, to always check for the validity and existence of suite versions and their callbacks before you use them.

Adobe After Effects

The Adobe After Effects application hosts Adobe Photoshop plug-ins as if it were Adobe Photoshop 3.0. This means that all the 3.0 API calls and functions are implemented, except:

1. Any callbacks related to Acquire or Export modules.

2. Any 3.0.4 callback services or suites.

Table 1: Adobe After Affects version and signature information

Description	Value
Mac OS version	3.0
Mac OS release date	12/20/95
Windows version	None
Windows release date	N/A
Backward-compatibility targets Mac, Win	3.0, N/A
Signature	'FXTC'

Table 2: Adobe After Effects emulating Adobe Photoshop host

Description	Value
Signature	'8BIM'
Host version support	3.0
Required adaptor	N/A
Resource	'PiPL'

Table 2: *(continued)*

Description	Value
Supported module types	Filter, Format, Parser
Plug-in folder default	Adobe After Effects/Plug-ins/ standard/ Photoshop Filters
Plug-in aliases	Automatically resolved by After Effects.
Plug-in load order	Loads references, but not code until execution request. Press *control-clear* to clear out the plug-in code cache, forcing the code to be reloaded.
How to access the different plug-ins while using Adobe After Effects:	
Filter modules	**Effects**»(sub-menu) "PS plugInName"=Normal filter "PS + plugInName"=Filter with 'ANIM' resource
Format modules	File»Import»"Footage"»"File type:"
Parser modules	Load at startup.

Creating dynamic resources

After Effects allows Photoshop plug-ins to be controlled over time. This is achieved through the addition of a simple resource called an 'ANIM'. ANIM resources are described in detail in the Adobe After Effects SDK.

In future columns, we'll continue to offer specific guidelines for different Adobe applications. §

Acrobat Column

Extracting Text from a PDF File

The proliferation of PDF files on the Web has encouraged developers to seek means of indexing the PDF files for search and retrieval. Ideally, the process would be as easy and transparent as indexing text and html files.

This month we will explore how to extract text from PDF files using a plug-in for the Acrobat® Exchange® program.

We should first understand how the text in a PDF file is generated, and the format of the PDF file contents. The PDF file creator is the ultimate arbiter of what streams of data representing content are placed first into a PDF file. The PDFWriter printer driver simply converts QuickDraw™ and GDI commands directly into corresponding PDF operators. If the creator app decides to output every third word on a page first, then go back and output the rest of the words, the PDFWriter will represent the content inside the PDF file in that order, too. Since PDF's goal was document visual integrity and not tagged structure, there is no effort made to collate the individual words into sentences. When

the PDF file is viewed, it is the x-y word locations that determine where the words are placed on the page.

In order to extract text from the PDF file, you must first create a **PDWordFinder** object. The **PDWordFinder** object specifies the font encodings and character information used for output text. The **PDWordFinder** also allows you to specify strings to be used for the ligature information, and what type of character each glyph represents, such as control codes, letter, uppercase, etc. Constants have been defined for these character types and more, and are listed in Declarations section of Technical Note #5168, "Acrobat Viewer Plug-in API On-line Reference." The API procedure declaration to create a **PDWordFinder** object is:

```
PDWordFinder PDDocCreateWordFinder (PDDoc doc, Uns16 *outEncInfo,
char **outEncVec, char **ligatureTbl, Int16 algVersion,
Uns16 rdFlags, void *clientData);
```

You can pass **NULL** for the arguments **outEncInfo**, **outEncVec**, and **ligatureTbl** to specify that the extracted text be in the default platform encoding. The default platform encodings are specified in Appendix C of the document "Portable Document Format Reference Manual."

The **rdFlags** argument specifies on the way the text is to be extracted from the PDF file. There are currently two ways to extract the text from a PDF file. You can get a table containing all the words on a page using **PDWordFinderAcquireWordList( )**, or you can get each word on a page individually using **PDWordFinderEnumWords( )**.

If you call **PDWordFinderAcquireWordList**, you will get up to two word lists filled after making the call. You can specify the **WXE_PDF_ORDER** constant and get back a pointer to words on the page, in PDF order. This is the quickest operation for the viewer to perform. If you specify the **WXE_XY_SORT** constant, then in the subsequent call to **PDWordFinderAcquireWordList( )**, you will receive a pointer to a list of words in PDF order, just as you did when specifying the **WXE_PDF_ORDER**. You will also get a pointer to a table that contains indexes into the PDF-ordered word list that will specify the x-y order of the words on the page.

If you use **PDWordFinderEnumWords( )**, you must specify only one constant for the **rdFlags** argument. The constant will control the order in which words are given back to you. When using **PDWordFinderEnumWords( )**, you'll also need to create a **PDWordProc** callback. This callback is invoked

for each word accessed on a page. The callback has the prototype:

```
ACCB1 boolean ACCB2 PDWordProc (PDWordFinder wObj, PDWord wInfo,
       Int32 pgNum, void *clientData);
```

The callback gets the **PDWordFinder** object, and the page number, along with the current **PDWord** and any client data specified when calling **PDWordFinderEnumWords( )**.

The last argument to **PDDocCreateWordFinder( )** is currently unused by the Acrobat viewers; pass **NULL** for the last argument.

At this point, no matter what method you use, you'll have a **PDWord**. A **PDWord** is an opaque data type that encapsulates the character representation of the string, the word style (color, name, etc.), and the location of the word on the page.

What next? Well, you can get a character representation of the word using **PDWordFilterWord( )**, which is the routine that the Catalog program uses to index PDF files, or **PDWordFilterString( )**, which was designed for known character sets like WinAnsi and MacRoman.

As an example, we'll pretend to be part of a company that wants to be able to index the text of PDF files. We'll put the above functions together in a simple routine to extract words to a text file. The complete code is shown in listing 1.

The plug-in was designed to be invoked from a menu item selection. The **ExecuteProc** named **TemplateCommand( )** is a good place to create the **PDWordFinder** since we can manage the creation and destruction of the **PDWordFinder** object from this single routine and eliminate the possibility of memory leaks.

The following line of code in **TemplateCommand( )** creates the **PDWordFinder** object. I'm only interested in the default encodings for the platform I'm performing the extraction on. Also, specifying word order of **WXE_PDF_ORDER** is fine, since the extracted text is not meant to be read like a text file, but only to be indicative of all words in the file.

```
PDWordFinder wf = PDDocCreateWordFinder(
      pddoc,
      NULL,
      NULL,
      NULL,
      WF_LATEST_VERSION,
      WXE_PDF_ORDER,
      NULL);
```

Now that the **PDWordFinder** object is created, we need to pass it as a parameter to one of the word extraction methods. We'll use **PDWordFinderEnumWords( )**.

```
PDWordFinderEnumWords (
        wf,
        pg,
        ASCallbackCreateProto(PDWordProc, enum_words_proc),
        NULL);
```

The **page** argument is the current page. You could easily extend this to iterate through all the pages of the document. I've passed a function named **enum_words_proc( )** that will process the words as called. The callback uses the **PDWordFilterString( )** to return a C-style string that can then be indexed.

Be sure to destroy the **PDWordFinder** object when finished using it by calling **PDWordFinderDestroy( )**.

Adobe offers another approach for companies wishing to extract the text from PDF files without using the Acrobat Exchange program or a plug-in. This approach is the Acrobat Toolkit.

The Acrobat Toolkit is a separate product that can be licensed through the Adobe Developers Association. The toolkit is a subset of the normal Acrobat Exchange program API that provides a library

which can be statically or dynamically linked to your application to provide text extraction facilities on UNIX and 32-bit Windows platforms.

You can learn more about the API calls discussed in this article by looking at Technical Note #5168, located on the Acrobat SDK. The Acrobat SDK also has an example plug-in named WordFind that provides a more extensive look at the **PDWordFinder** and **PDWord** options. You can also explore getting more detailed information about the actual words by looking at the **PDWord** object API calls.

Listing 1 *(a text version of this source code snippet is included: V5NO6L01.TXT)*

```
#include "ASCalls.h"
#include "AVCalls.h"
#include "PDCalls.h"ACCB1 boolean ACCB2 enum_words_proc (PDWordFinder wObj, PDWord wInfo,
       Int32 pgNum, void *clientData)
{
```

```
        DURING
                Uns8 length = PDWordGetLength(wInfo);
                char* cstr = (char*)ASmalloc(length+1);
                PDWordGetString(wInfo, cstr, length+1);
                /* save the word to file */
                AVAlertNote(cstr);
        HANDLER
                return false; /* failure */
        END_HANDLER
        return true;
}

/* Do our command action */
static ACCB1 void ACCB2 TemplateCommand(void* data)
{
        PDDoc pddoc = AVDocGetPDDoc(AVAppGetActiveDoc());
        AVPageView pgview = AVDocGetPageView(AVAppGetActiveDoc());
        Int32 pg = AVPageViewGetPageNum(pgview);
```

```
DURING
PDWordFinder wf = PDDocCreateWordFinder(
       pddoc,
       NULL,
       NULL,
       NULL,
       WF_LATEST_VERSION,
       WXE_PDF_ORDER,
       NULL);

PDWordFinderEnumWords (
       wf,
       pg,
       ASCallbackCreateProto(PDWordProc, enum_words_proc),
       NULL);
```

```
        HANDLER

        END_HANDLER
        PDWordFinderDestroy(wf);
}


/*
** InitProc
*/
static ACCB1 ASBool ACCB2 MyInit(void)
{
        AVMenubar menubar = AVAppGetMenubar();
        AVMenu commandMenu;
        AVMenuItem menuItem;        /* Set up our command menuitem. */
        commandMenu = AVMenubarAcquireMenuByName(menubar, "Extensions");
        if (commandMenu) {
```

```
        /* Add menu item to submenu */
                menuItem = AVMenuItemNew("SimpleTextX", "ADBE:SimpleTextX",
NULL, false, NO_SHORTCUT, 0, NULL, gExtensionID);
                AVMenuItemSetExecuteProc(
                        menuItem,
                        ASCallbackCreateProto(AVExecuteProc, &TemplateCommand),
                        NULL);
                AVMenuAddMenuItem(commandMenu, menuItem, APPEND_MENUITEM);
        }

        return true;
}
```

```
/*
** UnloadProc
*/
static ACCB1 ASBool ACCB2 MyUnload(void)
{
       return true;
}

/*
** PIHandshake
** Required Plug-in handshaking routine: Do not change its name!
*/
ACCB1 ASBool ACCB2 PIHandshake(Uns32 handshakeVersion, void *handshakeData)
{
```

```
        if (handshakeVersion == HANDSHAKE_V0200) {
                PIHandshakeData_V0200 *hsData = (PIHandshakeData_V0200 *)handshakeData;
                hsData->extensionName = ASAtomFromString("ADBE_SimpleTextX");
                hsData->exportHFTsCallback = NULL;
                hsData->importReplaceAndRegisterCallback = NULL;
                hsData->initCallback = ASCallbackCreateProto(PIInitProcType, MyInit);
                hsData->unloadCallback = ASCallbackCreateProto(PIUnloadProcType, MyUnload);
                return true;
        }
        return false;
}
```

Colophon

All proofs and final output for this newsletter were produced using Adobe PostScript® and Adobe Acrobat Distiller® for final file output. The document review process was accomplished via electronic distribution using Adobe Acrobat software.

Managing Editor:
Jennifer Cohan

Technical Editor:
Nicole Frees, Susan Tiner

Art Director:
Min Wang

Designer:
Lorsen Koo

Contributors:
Brian Andrews, Andrew Coven, Ben Arbogast

This newsletter was created using Adobe Acrobat, Adobe PageMaker, and Adobe Photoshop, and font software from the Adobe Type Library.