Adobe Developers Association

ADA news

In This Issue

Volume 4, Number 6

Adobe Photoshop Plug–in Properties

When Adobe Photoshop™ 3.0 was released, it contained a wealth of new features including the very powerful layers feature. Users quickly grasped the capabilities offered by layers, and soon viewed layers as an indispensible part of Adobe Photoshop. After using version 3.0, it's hard to imagine going back to an earlier version.

What's less visible to users, but just as important, are new features of the Adobe Photoshop plug-in application programmer's interface (API). Architectural enhancements in the 3.0 plug-in interface provide greater control over how plug-in modules work with the application. We'll talk about one of these new features in detail in this article—plug-in properties.

A *plug-in property* provides information about the plug-in module it belongs to. Properties contain metadata which Adobe Photoshop uses to:

1. identify the type and name of the plug-in,
2. determine whether the plug-in is compatible with the host application,
3. run the optimal version of a plug-in module's code, depending on the platform Adobe Photoshop is running on, and
4. control the conditions under which a plug-in module is available to the user.

Plug-in properties are easily extensible; other plug-in hosts can define application-specific or plug-in type-specific extensions using properties.

Properties are grouped into a data structure known as a *Plug-in Property List*, or *PiPL* (pronounced "pipple"). A PiPL contains all the information Adobe Photoshop needs to load and run a plug-in module. PiPLs replace the PiMI ("pimmy") resources used in earlier versions of Adobe Photoshop.

Usually, a plug-in file contains only one plug-in module. There are situations where it makes sense to group two or more plug-in modules into a single file (matching acquire and export modules, for example). You can do this under Adobe Photoshop 3.0 by creating multiple PiPLs, one for each plug-in module in the file. Each PiPL can be customized for the corresponding plug-in module, offering flexibility to the developer.

Creating and Editing PiPLs

Under Mac® OS, PiPLs are stored in Macintosh® resources of type 'PiPL'. Similarly under Windows®, PiPLs are compiled into Windows resources. The precise structure of plug-in properties and PiPLs are described in detail in the *Photoshop 3.0 SDK Guide*.

Most often you will create PiPLs using the Macintosh Rez source code language. Under Mac OS, you then compile the PiPL resource using Rez (MPW® or Metrowerks) or SARez (Symantec).

You can also create and edit PiPLs using a graphical resource editor like ResEdit™ or Resorcerer. Unfortunately, ResEdit can only edit PiPL resources as raw hex bytes; the TMPL format is

continued on page 2

Adobe

How To Reach Us

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 24 hours.

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.
devsup-person@mv.us.adobe.com

Europe:
eurosupport@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:
European Engineering
Support Group
Adobe Systems Benelux B.V.
P.O. Box 22750
1100 DG Amsterdam,
The Netherlands

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

Adobe Photoshop Plug–in Properties

inadequate for expressing the structure of PiPL resources. This limits the usefulness of ResEdit for this purpose.

The Windows version of the SDK includes the *CNVTPIPL.EXE* utility to convert Macintosh Rez source code into Windows .RC format, which can be compiled by Visual C++.™ Although you can create PiPL resources directly in the Visual C++ resource editor, this has two drawbacks. First, if your plug-in is intended to work on both Mac and Windows, keeping your PiPL source in Rez format allows you to maintain a single source for both platforms. Even if you are only developing for the Windows platform, however, creating your PiPL resource in Rez format and converting using *CNVTPIPL.EXE* will reduce the likelihood of errors, since *CNVTPIPL.EXE* handles byte ordering and padding issues automatically.

The Clouds Plug-in Filter

Let's look at the PiPL used by one of the new plug-in modules in Adobe Photoshop 3.0: the Clouds filter. This plug-in is found under the Filters→Render menu, and generates a random cloud pattern. (If you aren't familiar with this plug-in, you may want to take a break from this heavy reading and play with it for a while. It's very relaxing.)

First, here's the entire PiPL resource for Clouds in Rez source code format. We will examine the parts of this file in more detail next.

```
/*
   File: Clouds.r
   Copyright 1992-94 by Adobe Systems, Inc.

*/

resource 'PiPL' (16000, purgeable)
{{

   /* General properties */

   Kind { Filter },
   Name { "Clouds" },
   Category { "Render" },
   RequiredHost { '    ' },
   Version { (latestFilterVersion << 16) | latestFilterSubVersion },

  /* Code descriptor properties */

   #if MSWindows
   CodeWin32X86 { "PLAIN_CLOUDS_ENTRY" },
   #else
   Code68K { '8BFM', $$ID },
   CodePowerPC { 0, 0, "PLAIN_CLOUDS_ENTRY" },
   #endif
```

Adobe Photoshop Plug–in Properties

```
   /* Image related properties */

   SupportedModes { noBitmap,
                    doesSupportGrayScale,
                    noIndexedColor,
                    doesSupportRGBColor,
                    doesSupportCMYKColor,
                    doesSupportHSLColor,
                    doesSupportHSBColor,
                    doesSupportMultichannel,
                    doesSupportDuotone,
                    doesSupportLABColor },

   /* Filter-specific properties */

   FilterCaseInfo
   {{
      /* Flat data, no selection */
      inStraightData, outStraightData, 0, 0, doNotCopySourceToDestination,

      /* Flat data with selection */
      inStraightData, outStraightData, 0, 0, doNotCopySourceToDestination,

      /* Floating selection */
      inStraightData, outStraightData, 0, 0, doNotCopySourceToDestination,

      /* Editable transparency, no selection */
      inStraightData, outFillMask, doesNotFilterLayerMasks,
         worksWithBlankData, doNotCopySourceToDestination,

      /* Editable transparency, with selection */
      inStraightData, outFillMask, doesNotFilterLayerMasks,
         worksWithBlankData, doNotCopySourceToDestination,

      /* Preserved transparency, no selection */
      inStraightData, outStraightData, 0, 0, doNotCopySourceToDestination,

      /* Preserved transparency, with selection */
      inStraightData, outStraightData, 0, 0, doNotCopySourceToDestination
   }}
}};
```

The first five properties in the PiPL resource provide basic information about the plug-in.

The *Kind* property indicates what type of plug-in this is, which determines which menu the plug-in will appear under, as well as how it is called by the plug-in host. The *Name* property provides the menu name and the optional *Category* property indicates that "Clouds" should appear under the "Render" submenu.

The *Version* and *RequiredHost* properties indicate to Adobe Photoshop what version of plug-in API this plug-in was created for, and whether Photoshop is the intended host application for this plug-in. The default host ID of four spaces is a wildcard, and matches any host.

continued on page 4

Adobe Photoshop Plug–in Properties

Code Descriptor Properties

Next are the code descriptor properties. They indicate what systems the plug-in will work with. The Clouds plug-in is designed for both Macintosh and Windows, and contains several code descriptor properties enclosed by `#ifdef` pre-processor statements.

Under Windows, the *CodeWin32X86* property indicates that the plug-in code is a 32-bit dynamic link library (DLL), and the entry point name for the plug-in code is "`PLAIN_CLOUDS_ENTRY`".

Fat Plug-ins and Really Fat Plug-ins

Under Mac OS, the *Code68K* and *CodePowerPC* properties indicate to Adobe Photoshop that this is a fat plug-in. The parameters in the *Code68K* property declaration indicate the resource type ('8BFM') and the ID ($$ID indicates the ID number of the PiPL resource, 16000) of the 680x0 code resource.

When run on a Power Macintosh®, Photoshop will load and execute the PowerPC™ code, which is stored in the data fork and managed by the Code Fragment Manager. The *CodePowerPC* property parameters indicate the offset within the data fork (0), the length of the plug-in code (0, which indicates that the fragment extends to the end of the file), and the entry point name, "`PLAIN_CLOUDS_ENTRY`". In Adobe Photoshop 3.0, you can put more than one plug-in module within a single file, and provide multiple PiPL resources containing *CodePowerPC* properties to identify their entry points. (Similarly, those PiPLs may contain *Code68K* properties pointing to different code resources.)

There is one more code descriptor not shown here, the *Code68KFPU* property. This property allows you to provide two 680x0 code resources, one compiled with the floating point hardware option enabled and the other without. Thus, a "really fat" plug-in contains three sets of code: PowerPC, 680x0, and 680x0 FPU.

If you include both *Code68K* and *Code68KFPU* properties in your PiPL (and the corresponding code resources compiled with the correct flags), Adobe Photoshop will load the right code resource depending on the user's hardware configuration. If you only include a *Code68KFPU* property, the user cannot select your plug-in when running on a 680x0 Macintosh without an FPU.

One word of caution with respect to fat plug-ins. The *Code68K, Code68KFPU,* and *CodePowerPC* properties provide information about what plug-in code is available in a plug-in module. Mac OS, however, uses a 'cfrg' (code fragment) resource to indicate the presence of PowerPC code in the data fork of a file. The problem is that if a user runs a program that strips out a fat file into just a 680x0-only file using a utility program, the plug-in can get into an inconsistent state; the PiPL resource indicates that a PowerPC version of the plug-in code is present, but the code is not there. Some compilers, including Metrowerks CodeWarrior,™ automatically

Attention: *Developers!*

Each month, we reserve space in our newsletter for input from our developers. Our Developers Column is a place for members of the ADA to share information with other developers.

If you would like to contribute to our Developers Column, please contact us with your ideas. We look forward to hearing from you.

Adobe Photoshop Plug–in Properties

generate a 'cfrg' resource. To prevent the problem of fat stripping, you can manually remove the 'cfrg' resource from your finished plug-in module.

Image Related Properties

The *SupportedModes* property indicates to Adobe Photoshop what image types this plug-in can process. The Clouds plug-in will render clouds in all image modes except bitmap and indexed color. If the current document in Photoshop is set to one of these modes, the "Clouds" filter menu command will be grayed out.

The last property in the Clouds PiPL resource, *FilterCaseInfo,* indicates what sort of processing this filter can perform. This property is specific to filter plug-ins only; acquire, export, and format modules define their own type-specific properties.

The *FilterCaseInfo* property contains seven groups of five fields, indicating what processing the filter will do. The seven groups correspond to different cases in which the Clouds plug-in filter may be called. For example, the first case represents the situation where the filter is executed when the user is currently editing the background layer and there is no transparency data or selection. The next case represents the situation where there is an active selection. Refer to the *Photoshop SDK Guide* for more information about the *FilterCaseInfo* property.

Summary

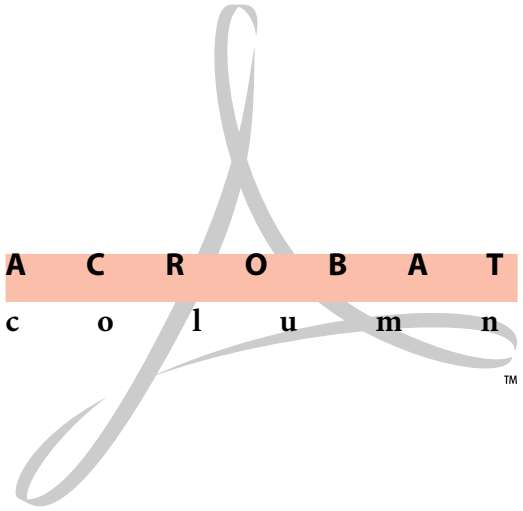
Plug-in properties provides greater control over how your plug-in module works. PiPLs contain property information across various levels:

- general *(Name, Category),*
- platform specific *(CodeWin32X86, CodePowerPC),*
- application specific *(SupportedModes),* and
- plug-in type specific *(FilterCaseInfo)*

Going forward, PiPLs will play an increasingly important role in Adobe's plug-in architecture.

Should you update your existing plug-in modules to the Photoshop 3.0 PiPL format? Yes, you can easily add a PiPL resource without changing any of your code. Depending on your plug-in's requirements, you can maintain backwards compatibility with Photoshop 2.5 while adding the benefits offered by PiPLs. You probably will want to at least recompile your plug-in, however, to take advantage of native PowerMac and 32-bit DLLs to achieve better performance.

The 3.0 API adds other new capabilities, including access to Photoshop color services and the `AdvanceStateProc`, which provides a more efficient way to handle large images. We'll talk about these more in a future newsletter. §



In this column, we'll describe the ways that plug-ins can enhance the security support in the Acrobat Exchange™ and Exchange LE viewers. Version 1.1 of the Portable Document Format, and version 2.0 of the Acrobat viewers, support file encryption and permissions. Encryption encodes a PDF file's contents to prevent unauthorized users from viewing it. Permissions specify the operations that authorized users can or cannot perform on a file. PDF permissions can disallow printing, copying text and graphics to the clipboard, adding notes, or making any modifications whatsoever to a document.

Plug-ins can change the way in which users are authorized to open files and can control the permissions granted to a user. They do this by providing a security handler, which is a group of functions that perform user authorization and grant permissions. The Acrobat viewers can have multiple security handlers installed simultaneously. Although plug-ins can change the authorization mechanism and control the granting of permissions, they cannot change the encryption algorithm used in PDF files— files are always encrypted using the RC4 algorithm from RSA Data Associates, Inc.

Acrobat Exchange ships with one security handler, named Standard, that supports user authorization via passwords. The Standard security handler supports two passwords— a user password that allows a file to be opened with owner-specified permissions, and an owner password that allows the permissions to be changed. Plug-ins can provide additional security handlers, which use other authorization mechanisms, for example, the presence of a specific file on disk, the presence of a particular hardware key, or a code entered via a magnetic card reader.

When saving a newly-secured file, the user must first choose the security handler for the document. The Acrobat viewer's Standard security handler is used by default, but users can choose any other available security handler by holding down the Option key (Macintosh) or Control key (Windows) when clicking on the "Security…" button in the "Save As…" dialog. Next, a function in the security handler is called to provide whatever data the security handler wishes to place in the PDF file for its use when the file is opened. Generally, this is information used by the function that authorizes the user, but it can be any information the security handler chooses. The Acrobat viewer automatically adds the security handler's name to the list of information written into the file, so that it can determine which security handler to invoke when the file is opened. Finally, another function in the security handler is called to provide the key that is used by the RC4 algorithm to encrypt the PDF file. Security handlers often calculate this key from the authorization data in some fashion, although they are free to calculate it in any way. After this, the PDF file is written out with all data (except for the information provided by the security handler) encrypted using the RC4 key provided by the security handler. The reason that the security handler's information is not RC4 encrypted is to avoid a circularity problem, in which the data needed to authorize a user (and calculate the decryption key) cannot be read until after the user has been authorized. The security handler is free, however, to encrypt its data in whatever way it chooses before passing it to the Acrobat viewer to append to the PDF file.

When a secured file is opened, the Acrobat viewer reads from the file the name of the security handler that secured the document. If that security handler is unavailable (because the relevant plug-in has not been loaded), the user is notified, and is not allowed to open the file. If the appropriate security

handler is available, a function in it is called to authorize the user, using whatever mechanism the security handler chooses. This function has access to the security-related information that was written into the file when it was saved. If the security handler does not authorize the user, for example because the user's machine lacks a necessary hardware key, then the file is not opened. If the security handler authorizes the user, the Acrobat viewer calls another function in the security handler to obtain the RC4 key, and uses the key to decrypt the file.

Security handlers also have a function that is called when a user attempts to change a file's permissions. This function decides which of the requested permissions to grant to the user. It generally bases the decision on some piece of authorization data. For example, the Standard security handler will only allow a user who has entered the owner password (not the user password) to change privileges in a file.

The Acrobat viewer's security handler mechanism provides a broad range of capabilities, allowing a plug-in to implement schemes such as hardware-based authorization, software-based authorization, or subscription systems. The Acrobat Plug-ins SDK, available through the ADA, contains all the information needed to write a security handler. §

Colophon

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:
Gail Blumberg

Designer:
Lorsen Koo

Contributors:
Tim Bienz, Paul Ferguson

Part Number ADA0059 9/95

Adobe PostScript