Adobe

# Adobe Graphics and Publishing

# Cross-Application Plug-in Development Resource Guide

**Version 1.2 Release 2**
**November 1996**

**Adobe Graphic Application Products**
**Cross-Application Plug-in Development Resource Guide**



Some of the material for this document was derived from earlier works by Thomas Knoll, Mark Hamburg and Zalman Stern. Additional contributions came from David Corboy, Kevin Johnston, Sean Parent and Seetha Narayanan. Information regarding specific SDKs, APIs, and product interfaces has been provided by Matt Foster, Brian Andrews, Paul Norton, and Paul Ferguson. This document was then compiled and edited by Andrew Coven.

# Version History

| Date | Author | Status |
|---|---|---|
| 6 February 1996 | Andrew Coven | First release |
| 6 March 1996 | Brian Andrews, Andrew Coven | Version 1.1. Update for Adobe After Effects 3. |
| 20 November 1996 | Andrew Coven | Version 1.2. Update for Adobe Photoshop 4.0. |

# 1. Introduction

Welcome to the Adobe Graphics Applications Plug-in Development Resource Guide!

This document is a guide to developing *plug–in modules* that operate in multiple applications. This includes Adobe After Effects, Adobe Illustrator, Adobe PageMaker, Adobe PhotoDeluxe, Adobe Photoshop, Adobe Premiere, and any other Adobe or third-party software that uses similar API structures.

## Audience

This guide is for C programmers who have written plug–ins for Adobe After Effects, Illustrator, PageMaker, Premiere, and Photoshop on Macintosh and Windows systems and wish to expand those plug-ins to operate in other applications besides their initial target application. This is called *cross-application plug-in development*. An example would be to expand a Photoshop Filter plug-in to operate in Illustrator, and PageMaker.

This guide assumes you are proficient in the C programming language and have worked in any or all of these development environments: Apple MPW; Metrowerks CodeWarrior Mac; Microsoft Visual C++; Windows NT; Windows 95.

You should have a working knowledge of the different Adobe products, and understand how plug–in modules work from a user's viewpoint. This guide assumes you understand terminology such as *paths*, *layers* and *masks*. For more information, consult the appropriate user's guide for your target products.

This guide does not contain information on creating plug-in modules for a specific application. Consult the individual *software development kits* for step-by-step instructions and example code.

## How to use this guide

This documentation is made to provide specific information on implementation and structuring issues for each major Adobe graphics application.

The best way to use this guide is to turn to the chapter containing specific information on the application that you wish to modify your plug–in to operate with.

If you writing plug–ins is new for you, we recommend you begin with the software development kit for the initial target application you wish to program your plug-in for, such as the *Adobe Photoshop SDK*.

Once you are familiar with plug-in terminology and the examples, consult this guide for different techniques when making your plug-in cross-application compatible.

### Under construction

This is the first release of this guide, and it is a work in progress. More detailed information about each product will be added as this document matures. Please report any errors or omissions to the Adobe Developers Association.

## GAP SDK tech notes mailing list

The Adobe Developers Association maintains a page on Adobe's world-wide-web site, `http://www.adobe.com`, which includes the latest SDK public releases and technical notes. You can also have the technical notes e-mailed to you directly by joining the Graphics Application Products SDK tech notes mailing list. The GAP SDK Tech Notes e-mail list is for Adobe After Effects, Adobe Illustrator, Adobe PageMaker, Adobe Photoshop and Adobe Premiere. Send an e-mail to

```
gap-dse@adobe.com
```

with the subject:

```
SUBSCRIBE GAP-SDK-TECH-NOTES
```

and these fields in your message body:

1. Your full name
2. Business name
3. Address
4. City
5. State
6. Country
7. Country code or Zip
8. Area code and phone number (business is fine)
9. ADA member number. "`N/A`" if not a member; "`Info`" if want info.
10. Any other e-mail addresses you want CC:'ed.

## About this guide

This programmer's guide is designed for readability on screen as well as in printed form. The page dimensions were chosen with this in mind. The Frutiger and Minion font families are used throughout the manual.

To print this manual from within Adobe Acrobat Reader, select the "Shrink to Fit" option in the Print dialog.

# 2. Getting Started

This chapter describes what plug–in modules are and provides information common to all plug–in modules. It introduces development strategies for creating plug-ins that are compatible with multiple applications.

## Plug–in modules and plug–in hosts

*Plug–in modules* are software programs developed by Adobe Systems and third–party vendors with Adobe Systems to extend an application. Plug–in modules can be added or updated independently by end users to customize Photoshop to their particular needs.

This guide also frequently refers to *plug–in hosts*. A plug–in host is responsible for loading plug–in modules into memory and calling them. The purpose of this guide is to assist in creating plug-in modules that operate under a variety of plug-in hosts.

These Adobe applications function as plug-in hosts: Adobe After Effects, Adobe Premiere, Adobe Illustrator, Adobe PageMaker, Adobe PhotoDeluxe, and Adobe Photoshop. All these applications support their own forms and types of plug-ins, which are detailed in their individual SDKs.

Most of these applications support some, but not all, Photoshop plug–in modules. Many applications from third–party developers support the use of Photoshop plug–in modules, as well.

Most plug–in hosts are application programs, but this not a requirement. A plug–in host may itself be a plug–in module. A good example of this is the “Photoshop Adapter” which allows Adobe Illustrator 6.0 to host Photoshop Format and Filter modules.

This guide is not designed for developers interested in creating plug–in hosts; the emphasis and goal for this guide is presenting information pertinent to creating plug–in modules.

Each plug-in host’s version will be listed when that particular application is discussed.

## Cross-development paradigm

Many developers have created plug-ins in their target application and want to expand their plug-in’s functionality to other applications. If you are going to take the time to make your plug-in compatible with one application, why not make it compatible with *all* of Adobe’s graphic application products? Adobe strongly encourages you to take the time to view all the details of this document, not just one section regarding one application.

An additional investment of 10-20% of your development time can result in a plug-in that operates in not just one application, but six (not counting third-party host applications.) We believe this is a lucritive return on your R&D investment.

## Version releases and compatibility issues

Designing your plug-in for multiple applications also makes it necessary to take into account different compatibility issues. Different hosts emulate

other hosts at different version levels. For instance, Adobe Premiere emulates Adobe Photoshop filter plug-ins as Photoshop version 2.5, while Adobe Illustrator emulates Adobe Photoshop filter plug-ins as Photoshop version 3.0.4.

Backward-compatibility means designing your plug-in to be accessible (and not just report an error message and quit) in earlier versions of applications. Table  lists the current versions of each piece of software and what version we recommend you target for backward-compatibility programming.

**Table 2-1: Version releases and compatibility chart**

| Application | Mac OS version | Mac OS release | Windows version | Windows release | Backward Mac, Win |
|---|---|---|---|---|---|
| Adobe After Effects | 3.1 | 2/1/96 | None | N/A | 3.0, N/A |
| Adobe Illustrator | 6.0 | 2/6/96 | 4.1 | 6/5/95 | 5.5, 4.1 |
| Adobe PageMaker | 6.0 | 6/1/95 | 6.0 | 8/1/95 | 5.0, 5.0 |
| Adobe PhotoDeluxe | 1.0 | 1/1/96 | None | N/A | 3.0, N/A |
| Adobe Photoshop | 4.0 | 11/18/96 | 4.0 | 11/18/96 | 2.5, 2.5 |
| Adobe Premiere | 4.2 | 8/1/95 | 4.0 | 5/1/95 | 4.0, 4.0 |

Forward-compatibility can be realized by programming your plug-ins with strict adherence to host signatures and suite version numbers. While it does require more programming when suites are not available, by religiously checking for host signatures and suite version numbers you can do a number of things by simply adding to your plug-in programming, as opposed to re-writing for every new version of a host that is released. Programming for backward- and forward-compatibility allows you to:

1. Take advantage of application-specific features
2. Program for backward- and forward- compatibility
3. Optimize for and use new suites as they become available.

## Cross-application plug-in development strategies

We recommend you follow this process for your cross-application plug-in development:

1. Assess and determine the problem your plug-in will solve.
2. Acquire the primary SDK for your base development.
3. Examine the examples and read the primary SDK.
4. Determine your development strategy for your base application.
5. Read the information in this guide with the needs of your plug-in in mind.
6. Reassess your development strategy for your base application.
7. Determine any host-requirements for the other target applications.
8. Program and create your plug-in.
9. Test under your base application.
10. Program and optimize based on testing results.
11. Test under the other target applications.

12. Modify and optimize based on those results.
13. Implement whatever beta-testing program you have.
14. Reassess and modify as needed.
15. Package and release your product.

# 3. Adobe After Effects

This chapter describes properties and useful resources of Adobe After Effects for creating plug-ins that work under multiple applications.

**Table 3-1: Adobe After Effects version and signature information**

| Description | Value |
|---|---|
| Mac OS version | 3.1 |
| Mac OS release date | 2/1/96 |
| Windows version | None |
| Windows release date | N/A |
| Backward-compatibility targets Mac, Win | 3.0, N/A |
| Signature | 'FXTC' |

# Adobe After Effects and Adobe Photoshop

**Table 3-2: Adobe After Effects emulating Adobe Photoshop host**

| Description | Value |
|---|---|
| Signature | '8BIM' |
| Host version support | 3.0, 3.0 |
| Required adaptor | N/A |
| Resource | 'PiPL' |
| 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)*<br>"PS *plugInName*"=Normal filter<br>"PS + *plugInName*"=Filter with 'ANIM' resource |
| Format modules | **File**»**Import**»"Footage"»"File type:" |
| Parser modules | Load at startup. |

### Host version support

Adobe After Effects emulates the Photoshop 3.0 Plug-in API. All of 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.

**Note:** Because hosts like Adobe After Effects implement a version of the Photoshop plug-in API that is earlier then the current version, it is very important you check for validity and existence of suite versions and their callbacks *before* you use them.

### 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 properties are detailed in the next chapter.

# 4. Adobe After Effects PiPLs

A Plug–In Property List, called a 'PiPL' (pronounced “pipple”) is a flexible, extensible data structure for representing a plug–in module’s metadata.

PiPLs contain all the information Adobe After Effects needs to identify and load plug–in modules, as well as flags and other static properties that control the operation of each plug–in. Your plug–in module should contain one or more 'PiPL' structures.

## Property structures and property lists

Plug–in *property structures* (or *properties*) are the basic units of information stored in a *property list*. Properties are variable length data structures, which are uniquely identified by a vendor code, property key, and ID number. The valid properties and formal grammar are documented later in this chapter.

## Creating PiPL resources

Under the Mac OS, PiPLs are stored as Macintosh resources. Under Windows, PiPLs are stored as Windows resources.

On the Macintosh, you can create and edit PiPL resources with a text editor and the *Rez* compiler, or you can use a graphical resource editor like *Resorcerer*. ResEdit cannot edit PiPL resources. If you are unfamiliar with the format of Rez source code, refer to the appropriate Apple documentation.

## Loading PiPL resources

When Photoshop launches, it scans all plug–in files for 'PiPL' resources. Historically, each type of plug–in had its own file type.

File types are only a matter of convention for 'PiPL' based plug–in modules. All known plug-in file types are searched for 'PiPL' resources and for those that are found, the information contained therein is used to determine the type of plug–in, code location, etc.

## Plug–in property lists

The plug–in property list structure has a version number and a count followed by one or more property structures.

```
typedef struct PIPropertyList
{
    int32                version;
    int32                count;
    PIProperty           properties[1];
} PIPropertyList;
```

**Table 4-1: PIPropertyList structure**

| Type | Field | Description |
|---|---|---|
| int32 | version | Current version is 0. |
| int32 | count | Number of properties in the 'PiPL'. 0=no properties. |
| PIProperty | properties | A variable length array of property data structures. |

## Plug–in properties

Each property has a vendor code, a key, an ID, and a length field.

```
typedef struct PIProperty
{
    OSType              vendorID;
    OSType              propertyKey;
    int32               propertyID;
    int32               propertyLength;
    char                propertyData[1];
    /* Implicitly aligned to multiple of 4 bytes. */
} PIProperty;
```

**Table 4-2: PIProperty structure**

| Type | Field | Description |
|---|---|---|
| OSType | vendorID | The vendor defining this property type. This allows other vendors to define their own properties in a way that does not conflict with either Adobe or other vendors. It is recommended that a registered application creator code be used for the `vendorID` to ensure uniqueness. After Effects creator code is 'FXTC' but all After Effects plug-ins use Adobe Photoshop's `vendorID` '8BIM'. |
| OSType | propertyKey | Property type, detailed in table 4-4. |
| int32 | propertyID | =0. Used to store more than one property of a given type. Reserved for future use. |
| int32 | propertyLength | Length of `propertyData`. Does not include any padding bytes to achieve four byte alignment. May be zero. |
| variable | propertyData | Variable length field containing contents of this property. Any values may be contained. |

## Adobe After Effects properties in the Mac OS and Windows

Specific properties can be extended in an upwardly compatible fashion by adding extra data at their end. The length field will allow an application to determine how much data is present, so optional properties can be omitted without concern. This is different from a fixed length structure where omitted fields must be given a default value.

It is intended for `PiPLs` to collect all plug-in metadata in a single place. this is useful for cross-platform development, since Windows lacks a resource management mechanism.

The 'PiPL' format is fairly portable in that everything is four byte aligned. All `OSType` and `int32` fields are represented in native byte order for a given platform so bytes of informationally indential `PiPLs` will differ between big-endian machines that run the Mac OS, and little-endian machines running Windows. The bytes of the `PiPL` section of a Windows binary resource are identical, but reversed, to the same resource in the Mac OS. This should not be of too much concern. As long as you use the pre-defined plug-in data types (table 4-3), they will be interpreted and stored correctly.

**Note:** An undefined `OSType` will not be converted automatically. It is normally interpreted as a `long` and you must supply the chars in reverse order for Windows implementation.

The After Effects API byte order is always big-endian.

## Adobe After Effects Basic data types

The following types are used to define properties:

**Table 4-3: Adobe After Effects Basic data types**

| Name | Description |
|---|---|
| int16, int32 | 16 and 32 bit integers. Stored in native byte order. |
| long | Same as `int32`. |
| short | Same as `int16`. |
| OSType | Same as `int32`. Typically denotes Mac OS 4 character filetypes like `'PiPL'`. |
| PString | Pascal style string where byte 1=length and content bytes follow. |
| CString | C style string where the content bytes are terminated by `NULL`. |
| Structures | Represented as would be in memory on the target platform. Native padding and alignment constraints are observed. |
| Arrays | Represented as a contiguous set of entries in the `'PiPL'` with native padding and alignment constraints observed. |
| ANIM_Float64 | Double. 8-byte IEEE 7 5 4. |

## General properties

These properties are common to all types of plug–in modules. The names of the properties (such as “PIKindProperty”) are the same as the #define names for the corresponding property keys.

**Table 4-4: Adobe After Effects general property keys**

| Type | Name | Key | Description |
|---|---|---|---|
| OStype | PIKindProperty | 0x6b696e64L ('kind') | Type or kind of plug-in.<br>'eFST'=Adobe After Effects Accelerator<br>'eFKT'=Adobe After Effects Effect<br>'FXIF'=Adobe After Effects I/O Format<br>'ARPI'=Adobe Illustrator<br>'8BXM'=Adobe Photoshop Accelerator<br>'8BAM'=Adobe Photoshop Acquire<br>'8BEM'=Adobe Photoshop Export<br>'8BFM'=Adobe Photoshop Filter<br>'8BIF'=Adobe Photoshop Format<br>'8BYM'=Adobe Photoshop Parser |
| PString | PINameProperty | 0x6e616d65L ('name') | Plug-in menu name for module in PICategoryProperty sub-menu. |
| PString | PICategoryProperty | 0x63617467L ('catg') | In the **Effects** menu, what sub–menu to list this plug–in. |
| int32 | PIVersionProperty | 0x76657273L ('vers') | Major and minor version number indicating which revision of the plug–in interface this plug–in was written for. The major version number indicates incompatible changes while the minor version number indicates incremental enhancements. The major version number is encoded in the most significant 16 bits of the 32 bit version number, the minor version number is encoded in the least significant 16 bits.<br>There are separate version numbers for each kind of plug–in. The current version for a given kind of plug–in is defined by a preprocessor macro in the header file defining the interface for that plug–in type. |
| int16 | PIPriorityProperty | 0x70727479L ('prty') | Plug-in load order. Also used to control the order in which items with the same name show up in menus.<br>Lower numbers (including negative ones) load first. If NULL, the default is zero. |
| FlagSet | PIImageModesProperty | 0x6d6f6465L ('mode') | Which image modes the plug–in supports. Adobe Photoshop, has 11 modes: bitmap, grayscale, indexed, RGB, CMYK, HSL, HSB, multi–channel, duotone, Lab, gray 16, and RGB 48.<br>This property determines whether your plug–in will be active (black) or inactive (gray) in Photoshop’s menus based on the current document’s image mode. |
| OSType | PIRequiredHostProperty | 0x686f7374L ('host') | Creator code of required host, such as '8BIM' for Adobe Photoshop. |

## Code descriptor properties

Code descriptors tell Adobe After Effects the type and location of a plug–in's code. More than one code descriptor may be included to build a "fat" plug–in which will run on different types of machines. After Effects will select the best performing option. After Effects makes sure that the callback structure is filled in with appropriate functions for the type of code that is loaded. So for PowerPC code, native function pointers will be provided and routine descriptor operations are not required either in calling the plug–in or for the plug–in to invoke callback functions.

**Table 4-5: Adobe Photoshop code descriptor properties**

| Type | Name | Key |
|---|---|---|
| PI68kCodeDesc | PI68KCodeProperty | 0x6d36386bL ('m68k') |
| This descriptor indicates a 68K code resource. The type for this property is:<br><br>`typedef struct PI68KCodeDesc`<br>`{`<br>`  OSType resourceType;`<br>`  int16 resourceID;`<br>`} PI68KCodeDesc;`<br><br>Any resource type may be used, but types of `PIKindProperty` from table 4-4 are strongly recommended. | | |
| PI68kCodeDesc | PI68KFPUCodeProperty | 0x36386670L ('68fp') |
| This descriptor is just like a `PI68KCodeDesc` except it will only be used on Macintosh machines that are equipped with FPU hardware. This allows vendors to easily ship plug–ins that take advantage of FPU hardware but still run on non–FPU Macs. | | |
| PICFMCodeDesc | PIPowerPCCodeProperty | 0x70777063L ('pwpc') |
| This descriptor indicates a PowerPC code fragment in the data fork of the plug-in file. The type for this property is as follows:<br><br>`typedef struct PICFMCodeDesc`<br>`{`<br>`  long fContainerOffset;`<br>`  long fContainerLength;`<br>`  char fEntryName[1];`<br>`} PICFMCodeDesc;`<br><br>Described in table 4-6. | | |

**Table 4-6: PICFMCodeDesc structure**

| Type | Field | Description |
|---|---|---|
| long | fContainerOffset | Data fork offset to the code fragment start. This allows more than one plug-in code fragment per file. |
| long | fContainerLength | Length of the code fragment. If the fragment extends to the end of the file or is the only fragment, the container length may be 0. |
| Pstring | fEntryName | Pascal string used to lookup the address of the function to call within the fragment. In order for the Code Fragment Manager to find an entrypoint by name, that name must be an exported symbol of the code fragment. If `NULL`, the default entrypoint will be used. `fEntryName` allows a single code fragment to contain more than one plug-in. |

## Filter–specific properties

These properties are applicable to Filter plug–in modules.

**Table 4-7: Adobe After Effects filter-specific properties**

| Length | Name | Key |
|---|---|---|
| 7 * 4-bytes | PIFilterCaseInfoProperty | 0x66696369L ('fici') |
| This key is for support for dynamically composited layers of image data.<br>A layer consists of color and transparency information for each pixel it contains. Previous versions did not have a transparency component. Completely transparent pixels have an undefined color. Filters will likely affect transparency data as well as color data. This is especially true for filters which produce spatial distortions.<br>The filter case info property allows flexibility in how transparency data is presented to filters. It controls the filtering process and presentation of data to the plug–in. This property provides information about what image data cases the plug–in supports. The current filtering situation is then compared to the supported cases and the best fitting case is choosen. The image data is then presented in that case. If none of the supported cases are usable, the filter will be disabled.<br>The case properties are an array of seven four byte entries, detailed in table 4-9. | | |

**Table 4-8: Filter cases**

| #define name | Description |
|---|---|
| 1=filterCaseFlatImageNoSelection | This is a background layer or a flat image. There is no transparency data or selection. |
| 2=filterCaseFlatImageWithSelection | No transparency data, but a selection may be present. The selection will be presented as mask data. |
| 3=filterCaseFloatingSelection | Image data with an accompanying mask. |
| 4=filterCaseEditableTransparencyNoSelection | Layer with transparency editing enabled and no selection. |
| 5=filterCaseEditableTransparencyWithSelection | Layer with transparency editing enabled and a selection. |
| 6=filterCaseProtectedTransparencyNoSelection | Layer with transparency editing disabled and no selection. |
| 7=filterCaseProtectedTransparencyWithSelection | Layer with transparency editing disabled and a selection. |

## FilterCaseInfo

Each of the 7 elements of the array contains a 4–byte `FilterCaseInfo`:

```
typedef struct FilterCaseInfo
{
    char inputHandling;
    char outputHandling;
    char flags1;
    char flags2;
} FilterCaseInfo;
```

### inputHandling & outputHandling

The *inputHandling* and *outputHandling* fields specify the pre–processing and post–processing actions on the image data respectively.

**Table 4-9: FilterCaseInfo handling modes**

| Handling mode | Description |
|---|---|
| 0=filterDataHandlingCantFilter | indicates that this case is not supported by the plug–in filter |
| 1=filterDataHandlingNone | indicates that the plug–in filter does not expect the plug–in host to do anything to the image data. |
| *The next three modes are matting cases, which are useful when performing distortions and blurs.*<br>You can matte the data, process it, and then dematte to remove the added color.<br>For these cases, the matting is defined as follows:<br>`mattedValue = ((unmattedValue * transparency) + 128) / 255 + ((matConstant * (255 - transparency)) + 128) / 255`<br>Dematting is defined as follows:<br>`unmattedValue = ((mattedValue - matConstant) ./ transparency) + matConstant`<br>with the `./` operator defined as an 8 bit fixed–point divide and the result value=0...255. | |
| 2=filterDataHandlingBlackMat | For the input case, matte the image data with black=0 values based on the transparency. For output, dematte the image data using black (=0) values. |
| 3=filterDataHandlingGrayMat | Matte the image data with gray (=128) values based on the transparency on input. Dematte the image data using gray values on output. |
| 4=filterDataHandlingWhiteMat | Matte the image data with white (=255) values based on the transparency on input. Dematte the image data using white values on output. |
| *Input-only related modes* | |
| 5=filterDataHandlingDefringe | Defringe transparent areas filling with the nearest defined pixels using taxicab distance. Note that this only applies to fully transparent pixels. |
| 6=filterDataHandlingBlackZap | Set color component of totally transparent pixels to black. |
| 7=filterDataHandlingGrayZap | Set color component of totally transparent pixels to gray. |
| 8=filterDataHandlingWhiteZap | Set color component of totally transparent pixels to white. |
| 10=filterDataHandlingBackgroundZap | Set color component of totally transparent pixels to the current background color. |

**Table 4-9: FilterCaseInfo handling modes (Continued)**

| Handling mode | Description |
|---|---|
| 11=filterDataHandlingForegroundZap | Set color component of totally transparent pixels to the current foreground color. |
| *Output-only related modes* | |
| 9=filterDataHandlingFillMask | This mode results in the transparency mask automatically being filled with full opacity in the area affected by the filter. This is only valid for the editable transparency cases. This option is provided to make it easy to write a plug–in similar to Photoshop's Clouds plug–in, which fills an area with a value. |

**Table 4-10: FilterCaseInfo flags1 parameters**

| Field | Description |
|---|---|
| 0=PIFilterDontCopyToDestinationBit | Normally source data is copied to the destination before filtering. This degrades performance for filters which write all the output pixels. Setting this bit inhibits copying. |
| 1=PIFilterWorksWithBlankDataBit | Determines whether the filter will work on "blank" areas that are completely transparent. If not, an error message will be given when the filter is invoked on a blank area. This is only valid for the editable transparency case because that is the only case where you could create opacity—in the protected transparency case, you would be left with what you started with: completely blank data. |
| 2=PIFilterFiltersLayerMaskBit | In cases where transparency is editable, this flag determines if Layer Masks are filtered. (See the "Add Layer Mask" item in the Layers palette menu to create a layer mask.) Setting this bit adds the layer mask to the set of target channels if: transparency for the layer is editable (i.e., this is one of the editable transparency cases), the bit is set, and the layer mask is specified as being positioned relative to the layer rather than the image in Layer Mask Options. The distinction based on position is based on the assumption that layer relative masks are distorted with the layer; image relative masks are independent of the layer. |

**Note:** This field is not a `FlagSet`. The first bit, `PIFilterDontCopyToDestinationBit`, is in the least–significant bit of the flag byte.

**flags2**

The *flags2* field of `FilterCaseInfo` is reserved, and should be zero.

## ANIM-specific properties

These properties are applicable to filters that are animatable.

**Table 4-11: Adobe After Effects ANIM-specific properties**

| Length | Name | Key |
|---|---|---|
| 32 bytes | ANIM_FILT_INFO_PROP | 0x61464C54L ('aFLT') |
| After Effects animatable filter description information. This key is for support for After Effects animatable filters (ANIMs).<br>Each filter should have one 'aFLT' and an arbitrary number of 'aPAR' properties. The combination of these two keys allows aware hosts to provide to animate the filter. If the filter shouldn't be driven, set ANIM_FF_DONT_DRIVE=TRUE. See table 4-12. | | |
| variable | ANIM_PARAM_INFO_PROP | 0x61464C54L ('aPAR') |
| After Effects animatable filter parameter information. This key is for support for After Effects animatable filters (ANIMs).<br>Each filter should have one 'aFLT' and an arbitrary number of 'aPAR' properties. The combination of these two keys allows aware hosts to provide to animate the filter.<br>The total number of 'aPAR' properties is included in the 'aFLT'. An 'aPAR' is distinguished but its PiPL ID, which progress from 0 to (number of parameters - 1). The order of the 'aPAR' properties implicitly reflects the order of the params in the filter's parameter block. See table 4-14. | | |

The ANIM_FilterDescription struct defines the After Effects animatable filter description and parameter information:

```
typedef struct ANIM_FilterDescription
{
    ANIM_FilterInfo   info;
    ANIM_ParamAtom    params[1];
} ANIM_FilterDescription, *ANIM_FilterDescriptionPtr,
**ANIM_FilterDescriptionH
```

### 'aFLT' property and ANIM_FilterInfo

The 'aFLT' property is described by the ANIM_FilterInfo struct:

```
typedef struct ANIM_FilterInfo
{
    long              spec_version_major;
    long              spec_version_minor;
    long              filter_params_version;
    ANIM_FilterFlags  flags;
    long              num_params;
    char              match_name[32];
    long              reserved[4];
} ANIM_FilterInfo;
```

**Table 4-12: ANIM_FilterInfo structure**

| Type | Field | Description |
|---|---|---|
| long | ANIM_MAJOR_VERSION | =1. Major version number. |
| long | ANIM_MINOR_VERSION | =0. Minor version number. |
| long | filter_params_version | This version will be stored to disk with the params. The params will be discarded if it's different then the ANIM version of the existing filter. |

**Table 4-12: ANIM_FilterInfo structure (Continued)**

| Type | Field | Description |
|---|---|---|
| ANIM_FilterFlags | flags | Filter flags. See table 4-13. |
| long | num_params | Number of parameters. |
| char | match_name | Cstring. Host will save this name to disk and use it to match when loading from disk. |
| long | reserved[4] | Reserved for future use. Set to zero. |

**Table 4-13: ANIM_FilterFlags structure**

| Field | Description |
|---|---|
| 0=ANIM_FF_HAS_RANDOMNESS | Same parameters and source does not produce exact same results. |
| 1=ANIM_FF_NON_GEOMETRIC | Pixel output depends on input pixel, not interpolation, exterpolation, or formula. |
| 2=ANIM_FF_FG_ANIMATABLE | Host should allow animation of foreground color. |
| 3=ANIM_FF_BG_ANIMATABLE | Host should allow animation of background color. |
| 4=ANIM_FF_PARAMS_IN_GLOBALS | Host should store globals according to filter specs. |
| 5=ANIM_FF_DIALOG_IN_RENDER | Filter inquiries user during `filterSelectorStart` or `filterSelectorContinue` instead of `filterSelectorParameters`. |
| 6=ANIM_FF_PARAMS_ARE_MAC_HANDLE | Parameters are stored as Macintosh handle. |
| 7=ANIM_FF_PARAMS_ARE_HANDLE | Parameters are stored as ANSI handle. |
| 8=ANIM_FF_PARAMS_ARE_PTR | Parameters are stored as Pointers. |
| 9=ANIM_FF_DOESNT_NEED_DLOG | Dialog doesn't init anything; host may fill opaque data with zeros and non-opaque data with reasonable values. |
| 10=ANIM_FF_DONT_DRIVE_ME | Don't load plug-in. |
| 11=ANIM_FF_RESERVED0 | Reserved. |
| 12=ANIM_FF_RESERVED1 | Reserved. |
| 13-31=Reserved | Reserved. |

**Note:** This field is not a `FlagSet`. The first bit, ANIM_FF_HAS_RANDOMNESS, is in the least–significant bit of the flag byte.

## 'aPAR' property and ANIM_ParamAtom

The 'aPAR' property is described by the `ANIM_ParamAtom` struct:

```
typedef struct ANIM_ParamAtom
{
    char              external_name[32];
    long              id;
    ANIM_DataType     data_type;
    ANIM_UIType       ui_type;
    ANIM_Float64      valid_min;
    ANIM_Float64      valid_max;
    ANIM_Float64      ui_min;
    ANIM_Float64      ui_max
    ANIM_ParamFlags   flags;
    long              byte_size;
    long              reserved[4];
```

```
} ANIM_ParamAtom;
```

**Table 4-14: ANIM_ParamAtom structure**

| Type | Field | Description |
|---|---|---|
| char | external_name | Cstring. Can be localized. |
| long | id | locally unique ID for paramter. Not the PiPL ID. 0=Reserved; <0=Reserved for host use.<br>The host uses this field to match parameters stored to disk with those in the parameter handle. You may add or remove parameters to your plug-in without changing `filter_params_version`.<br>If you change this value in the future and `ANIM_FF_DONT_NEED_DLOG=FALSE`, old data may be discarded. |
| ANIM_DataType | data_type | If opaque, ignore below except byte_size. See table 4-15. |
| ANIM_UIType | ui_type | User interface type. See table 4-16. |
| ANIM_Float64 | valid_min | Used for slider. Set valid_min=valid_max=0 for full range. |
| ANIM_Float64 | valid_max | |
| ANIM_Float64 | ui_min | Used for slider. Set ui_min=ui_max=0 for full range. |
| ANIM_Float64 | ui_max | |
| ANIM_ParamFlags | flags | Parameter flags. See table 4-17. |
| long | byte_size | Byte size of parameter data. |
| long | reserved[4] | Reserved for future use. Set to zero. |

**Table 4-15: ANIM_DataType structure**

| Field | Description |
|---|---|
| 0=ANIM_DT_OPAQUE | Opaque. |
| 1=ANIM_DT_CHAR | Character. |
| 2=ANIM_DT_SHORT | Short integer. |
| 3=ANIM_DT_LONG | Long integer. |
| 4=ANIM_DT_UNSIGNED_CHAR | Unsigned character. |
| 5=ANIM_DT_UNSIGNED_SHORT | Unsigned short integer. |
| 6=ANIM_DT_UNISNGED_LONG | Unsigned long integer. |
| 7=ANIM_DT_FIXED | Fixed 16:16. |
| 8=ANIM_DT_UNSIGNED_FIXED | Fixed unsigned 16:16. |
| 9=ANIM_DT_EXTENDED_96 | 12 byte value. Not recommended. |
| 10=ANIM_DT_DOUBLE_64 | 8 byte IEEE 7 5 4. |
| 11=ANIM_DT_FLOAT_32 | 4 byte IEEE 7 5 4. |

**Table 4-16: ANIM_UIType structure**

| Size | Field | Description |
|---|---|---|
| 0 | 0=ANIM_UI_NO_UI | Still must have name and data type. If not opaque, will animate. |
| sizeof(data_type) | 1=ANIM_UI_ANGLE | Angle. |
| sizeof(data_type) | 2=ANIM_UI_SLIDER | Slider. |
| 2*sizeof(data_type) | 3=ANIM_UI_POINT | (h,v) Point. |

**Table 4-16: ANIM_UIType structure (Continued)**

| Size | Field | Description |
|---|---|---|
| 4*sizeof(data_type) | 4=ANIM_UI_RECT | (t,l,b,r) Rectangle. |
| 3*sizeof(data_type) | 5=ANIM_UI_COLOR_RGB | RGB Color space. |
| 4*sizeof(data_type) | 6=ANIM_UI_COLOR_CMYK | CMYK Color space. |
| 3*sizeof(data_type) | 7=ANIM_UI_COLOR_LAB | L*a*b Color space. |

**Table 4-17: ANIM_ParamFlags structure**

| Field | Description |
|---|---|
| 0=ANIM_PF_IS_RES_DEPENDENT | Boolean. TRUE=your plug-in can adjust parameters dynamically as host changes resolution. |
| 1=ANIM_PF_SPACE_IS_RELATIVE | For ANIM_UI_POINT and ANIM_UI_RECT only.<br>TRUE=Relative mode: 0.0=left or top; 1.0=right or bottom.<br>FALSE=Absolute mode: Value=pixels. |
| 2=ANIM_PF_RESTRICT_BOUNDS | For ANIM_UI_POINT and ANIM_UI_RECT only.<br>TRUE=Don't call filter when point or rect is outside bounds of source.<br>FALSE=Call filter with any input point or rect. |

**Note:** These fields are not FlagSets. For instance, the first bit of the ANIM_ParamFlags structure, ANIM_PF_IS_RES_DEPENDENT, is in the least–significant bit of the flag byte.

## Effect–specific properties

These properties are applicable to Effect plug–in modules.

**Table 4-18: Adobe After Effects effect-specific properties**

| Type | Name | Key | Description |
|---|---|---|---|
| 2 * short | AEPiPLVersion | 0x65505652L ('ePVR') | Major and sub-version of Adobe After Effects PiPL. Must be 2 and 0. |
| 2 * short | PF_PLUG_IN_VERSION | 0x65535652L ('eSVR') | Major and sub-version of Effect specification. Must be 11 and 0. |
| int32 | PF_Vers | 0x65564552L ('eVER') | Effect version. See table 4-19. |
| int32 | PF_Outflags | 0x65474c4fL ('eGLO') | Global flags for effect output. See table 4-20. |
| Cstring | AEEffectMatchName | 0x654d4e41L ('eMNA') | String with effect name. |

**Table 4-19: PF_Vers parameters**

| Field | Description |
|---|---|
| 0...8=PF_Version_BUILD | Build version number. |
| 9...10=PF_Version_STAGE | Stage of build by name.<br>0=Develop (PF_Stage_DEVELOP)<br>1=Alpha (PF_Stage_ALPHA)<br>2=Beta (PF_Stage_BETA)<br>3=Release (PF_Stage_RELEASE) |
| 11...14=PF_Version_BUGFIX | Version number of bug fix release. |
| 15...18=PF_Version_SUBVERS | Minor/sub-version number. |
| 19...21=PF_Version_VERS | Major version number. |
| 22...31=Reserved | Reserved. Must be zero. |

### PF_OutFlags

The Effect Global Outflags describe how your effect responds to many of the PF_Cmd sequence callbacks. Unless otherwise noted, you should set and send these flags at PF_Cmd_GLOBAL_SETUP.

**Table 4-20: PF_Outflags parameters**

| Field | Description |
|---|---|
| 0=PF_OutFlag_KEEP_RESOURCE_OPEN | Doesn't close effect when done; keeps resource fork and parameters available. |
| 1=PF_OutFlag_WIDE_TIME_INPUT | Effect requests information about a non-current time (such as the previous video frame). |
| 2=PF_OutFlag_NON_PARAM_VARY | Effect bases output on more than the param list. |
| 3=PF_OutFlag_SEND_PARAMS_UPDATE | Effect updates controls values after parameter changes. Ex: If you want a slider to display text descriptions of numberical values, specify this flag, then get PF_Cmd_PARAMS_UPDATE whenever the user alter parameters. |

**Table 4-20: PF_Outflags parameters (Continued)**

| Field | Description |
|---|---|
| 4=PF_OutFlag_SEQUENCE_DATA_NEEDS_FLATTENING | Sequence data handle contains other pointers or handles. For sequence data, you will be called with PF_Cmd_SEQUENCE_RESETUP. Store a boolean at a common offset in your unflattened and flattened data indicating whether it's flat or not. On PF_Cmd_SEQUENCE_RESETUP and flat=TRUE then you should unflatten the data, free the flattened data handle, and set sequence_data in PF_OutData.<br>If you set the data=NULL when you flatten it, you will not be sent PF_Cmd_SEQUENCE_RESETUP to unflatten. Instead, you may get a RENDER call with data=NULL. |
| 5=PF_OutFlag_I_DO_DIALOG | Effect responds to PF_Cmd_DO_DIALOG. |
| 6=PF_OutFlag_USE_OUTPUT_EXTENT | Effect only process or changes behavior based on visible-image-area rect; extent rect change should cause re-render. |
| 7=PF_OutFlag_SEND_DO_DIALOG | Effect requires options dialog box to be presented at least once. Set during PF_Cmd_SEQUENCE_SETUP. PF_Cmd_DO_DIALOG will be sent right after. |
| 8=PF_OutFlag_DISPLAY_ERROR_MESSAGE | If return_msg in PF_OutData is a string, the host will display it. TRUE=display string as error dialog; otherwise display string as generic dialog. |
| *These fields are new since version 2.0 of Adobe After Effects.* | |
| 9=PF_OutFlag_I_EXPAND_BUFFER | Set if you expand the effect buffers beyond the layer dimensions. |
| 10=PF_OutFlag_PIX_INDEPENDENT | Output of a given pixel is not dependent on the values of surrounding pixels. |
| 11=PF_OutFlag_I_WRITE_INPUT_BUFFER | Effect writes to the input buffer. Use with discretion: this is useful as a scratch buffer, but invalidates some host speedups in rendering. |
| 12=PF_OutFlag_I_SHRINK_BUFFER | Your effect can shrink its buffer based on the extent rect. Use for memory efficiency. |
| 13=PF_OutFlag_WORKS_IN_PLACE | TRUE=effect can use the same buffer for both input and output; otherwise requires separate buffers. |
| 14=PF_OutFlag_SQUARE_PIX_ONLY | Supports square pixels. Ignored. |
| 15=PF_OutFlag_CUSTOM_UI | Has custom user interface and will process PF_Cmd_EVENT. See AE_EffectUI.h. |
| 16=PF_OutFlag_CUSTOM_NTRP | Use custom interpolation. See PF_FlatCustomDef in AEEffect.h. |
| 17=PF_OutFlag_REFRESH_UI | If set, host will call plug-in with update UI event right before plug-in exits. |
| 18=PF_OutFlag_NOP_RENDER | Set during frame setup if current rendering won't affect the image and may be skipped. |
| 19=PF_OutFlag_I_USE_SHUTTER_ANGLE | Effect is based on shutter_angle field. |
| 20=PF_OutFlag_I_USE_AUDIO | Effect is based on audio values. See *audio callbacks* in the *After Effects SDK*. |

**Table 4-20: PF_Outflags parameters (Continued)**

| Field | Description |
|---|---|
| 21=PF_OutFlag_I_AM_OBSOLETE | Set if you want your plug-in to be available, but not appear in the **Effect** menu. |
| 22=PF_OutFlag_RESERVED1 | Reserved. Must be zero. |
| 23=PF_OutFlag_RESERVED2 | Reserved. Must be zero. |
| 24=PF_OutFlag_RESERVED3 | Reserved. Must be zero. |
| 25=PF_OutFlag_RESERVED4 | Reserved. Must be zero. |
| 26...31=Reserved | Reserved. Must be zero. |

**Note:** These fields are not `FlagSets`. For instance, the first bit of the `PF_Outflags` structure, `PF_OutFlag_KEEP_RESOURCE_OPEN`, is the least–significant bit of the flag byte.

## Format–specific properties

These properties are applicable to Format plug–in modules.

**Table 4-21: Adobe After Effects format-specific properties**

| Type | Name | Key | Description |
|---|---|---|---|
| TypeCreator-Pair | PIFmtFileTypeProperty | 0x666d5443L ('fmTC') | Default type and creator code used for files newly created with this format plug–in.<br>Under Windows, files don't store TypeCreator information, except internally, so this property is not required; they are always interpreted as of type 'BINA' and creator 'mdos'.<br>All the info regarding what files can be read and written is obtained from the PIReadExtProperty or the PIFilteredExtProperty.<br>Under Windows, PiMI extensions are converted to PIReadExtPropertys, so use of PIFilteredExtProperty requires additional coding if you are porting a 16–bit plug–in format module to 32–bit. |
| Array of TypeCreator-Pair | PIReadTypesProperty | 0x52645479L ('RdTy') | List of type and creator pairs which the format plug–in can read. Specifying a value of four spaces (0x20202020L) matches any type or creator. |
| Array of TypeCreator-Pair | PIFilteredTypesProperty | 0x66667454L ('fftT') | List of type and creator pairs for which the file format plug–in should be called to determine if the file can be read. Specifying a value of four spaces (0x20202020L) matches any type or creator. |
| Array of OSTypes | PIReadExtProperty | 0x52644578L ('RdEx') | List of extensions which the format plug–in can read. The extension is stored in the first three characters of the OSType. The fourth character must be a space. |
| Array of OSTypes | PIFilteredExtProperty | 0x66667445L ('fftE') | List of extensions for which the file format plug–in should be called to determine if the file can be read. |

**Table 4-21: Adobe After Effects format-specific properties (Continued)**

| Type | Name | Key | Description |
|---|---|---|---|
| FlagSet | PIFmtFlagsProperty | 0x666d7466L ('fmtf') | This property contains a set of flags which control the operation of file format plug–ins. The default value for any flag is FALSE. See table 4-22. |
| Point | PIFmtMaxSizeProperty | 0x6d78737aL ('mxsz') | The maximum number of rows and columns that can be in an image saved in this format. Photoshop will use this field to screen out ineligible formats. |
| Array of int16s | PIFmtMaxChannelsProperty | 0x6d786368L ('mxch') | An array of counts of the maximum number of channels which can/will be saved for a given image mode.<br>This array is indexed by the plug–in mode constants. For example, if your format plug–in supports a single alpha channel in RGB mode, you should set maxChannels [plugInModeRGBColor]=4.<br>A plug–in may still be asked to save more channels than it reports it can support. This field exists primarily so that Photoshop can warn the user that alpha channels will be discarded. |

**Table 4-22: PIFmtFlagsProperty parameters**

| Field | Description |
|---|---|
| 0=PIFmtReadsAllTypesFlag | Obsolete. |
| 1=PIFmtSavesImageResourcesFlag | Resources besides image data, such as printing information, pen tool paths, etc.. are known as image resources. The plug–in format has the option of taking responsibility for these resources by reading and writing a block of data containing the image resources. If FALSE, Photoshop will add the image resources to the file's Mac OS resource fork but this will not be portable to other platforms. |
| 2=PIFmtCanReadFlag | =TRUE if the file format can read files. |
| 3=PIFmtCanWriteFlag | =TRUE if the file format can write files. |
| 4=PIFmtCanWriteIfReadFlag | Whether plug–in can write the file if the plug–in originally read the file. |

# Input/output-specific properties

These properties are applicable to Input/Ouput Format plug–in modules.

**Table 4-23: Adobe After Effects format-specific properties**

| Type | Name | Key | Description |
|---|---|---|---|
| int32 | AEImageFormatExtensionInfo | 0x46584d46L ('FXMF') | Adobe After Effects Imageformat Extension Information. Describes dynamic resources of module. See table 4-24. |

## AEImageFormatExtensionInfo

```
typedef struct AEImageFormatExtensionInfo
{
    long              majorVersion;
    long              minorVersion;
    int32             extensionFlags;
    long              reserved;
    char              signature;
} AEImageFormatExtensionInfo;
```

**Table 4-24: AEImageFormatExtensionInfo structure**

| Type | Field | Description |
|---|---|---|
| long | majorVersion | Major version number. |
| long | minorVersion | Minor version number. |
| int32 | extensionFlags | Flags describing resource. See table 4-25. |
| long | Reserved. | Reserved. |
| char | signature | Cstring. Localizable name of plug-in. |

**Table 4-25: AEImageFormatExtensionInfo extensionFlags parameters**

| Field | Description |
|---|---|
| 0=Input | Input module present. |
| 1=Output | Output module present. |
| 2=File | Direct correspondence to filetype in file system. |
| 3=Still | Still image support (`Video=FALSE`). (`PICS` file format is an example of Video). |
| 4=Video | Video support (`Still=FALSE`) |
| 5=Framestore | Time independent frame store. If `TRUE`, `Still=TRUE`. |
| 6=InteractGet | User interaction required for new sequence. Required if `File=FALSE` and `Input=TRUE`. |
| 7=InteractPut | User interaction required for new output. Required if `File=FALSE` and `Output=TRUE`. |
| 8=InteractPutRevert | User interaction required for new output, even if `revertInfo` is available. |
| 9=NonSeqAddFrame | Add frame can handle non-sequential times. |
| 10=NoOutputDialog | Has no output options dialog. |
| 11...31=Reserved. | `Reserved. Must be zero.` |

## Adobe After Effects PiPL syntax

This information is included as reference material. If you use the example source code and the documentation included on the Adobe After Effects SDK, you probably won't need to worry about the specifics of the PiPL syntax.

```
# Miscellaneous definitions

<OSType>

<int16>

<int32>

<float64>

<epsilon> :=

# Beginning of real grammar.

<PiPL spec> := <resource header> <resource body>

<resource header> :=
    "resource" "'PiPL'" "("
     <resourceID> <optional resource name> <optional attribute list>
    ")"

<optional name> :=
    <epsilon> |
    "," <string>

<optional attribute list> :=
    <epsilon> |
    "," <attribute> <attribute list tail>

<attribute list tail> :=
    <epsilon> |
     "|" <attribute> <attribute list tail>

<resource body> :=
    "{" "{"
    <property list>
    "}" "}"

<property list tail> :=
    <epsilon> |
    "," <property> <property list tail>

<property list> :=
    <epsilon>
    | <property> <property list tail>

<property> :=
    <kind property> |
    <version property> |
    <priority property> |
    <required host property> |
    <name property> |
    <category property> |
    <68k code descriptor property> |
```

```
    <powerpc code descriptor property> |
    <win32 x86 code property> |
    <supported modes property> |
    <filter case info property> |
    <format file type property> |
    <read types property> |
    <write types property> |
    <filtered types property> |
    <read extensions property> |
    <write extensions property> |
    <filtered extensions property> |
    <format flags property> |
    <format maximum size property> |
    <format maximum channels property> |
    <parsable types property> |
    <parsable extensions property> |
    <filtered parsable types property> |
    <filtered parsable extensions property> |
    <parsable clipboard types property> |
    <animatable filter description>

<kind property> := "Kind" "{" <kind ID> "}"

<kind ID> := <OSType> |
    "Filter" |
    "Parser" |
    "ImageFormat" |
    "Extension" |
    "Acquire" |
    "Export"

<version property> := "Version" "{" <version clause> "}"

<version clause> := <int32> |
    "(" <wired version ID high> "<<" "16" ")" "|"
    "(" <wired version ID low> ")" |
    <wired version ID>

<wired version ID> := "FilterVersion" |
    "ParserVersion" |
    "ImageFormatVersion" |
    "ExtensionVersion" |
    "AcquireVersion" |
    "ExportVersion"

<wired version ID high> := "latestFilterVersion" |
    "latestParserVersion" |
    "latestImageFormatVersion" |
    "latestExtensionVersion" |
    "latestAcquireVersion" |
    "latestExportVersion"

<wired version ID high> := "latestFilterSubVersion" |
    "latestParserSubVersion" |
    "latestImageFormatSubVersion" |
    "latestExtensionSubVersion" |
    "latestAcquireSubVersion" |
    "latestExportSubVersion"

<priority property> := "Priority" "{" <int16> "}"

<required host property> := "Host" "{" <OSType> "}"
```

```
<name property> := "Name" "{" <string> "}"

<category property> := "Category" "{" <string> "}"

<68k code descriptor property> := "Code68k" "{" <OSType>, <int16> "}"

<powerpc code descriptor property> := "CodePowerPC" "{"
    <int32>, <int32> <optional name> "}"

<win32 x86 code property> := "CodeWin32X86" "{" <string> "}

<bitmap support> := "noBitmap" | "doesSupportBitmap"
<grayscale support> := "noGrayScale" | "doesSupportGrayScale"
<indexed support> := "noIndexedColor" | "doesSupportIndexedColor"
<RGB support> := "noRGBColor" | "doesSupportRGBColor"
<CMYK support> := "noCMYKColor" | "doesSupportCMYKColor"
<HSL support> := "noHSLColor" | "doesSupportHSLColor"
<HSB support> := "noHSBColor" | "doesSupportHSBColor"
<multichannel support> := "noMultichannel" | "doesSupportMultichannel"
<duotone support> := "noDuotone" | "doesSupportDuotone"
<LAB support> := "noLABColor" | "doesSupportLABColor"

<supported modes property> := "SupportedModes"
    "{"
    <bitmap support> ","
    <grayscale support> ","
    <indexed support> ","
    <RGB support> ","
    <CMYK support> ","
    <HSL support> ","
    <HSB support> ","
    <multichannel support> ","
    <duotone support> ","
    <LAB support>
    "}"

<filter case info property> := "FilterCaseInfo"
    "{"
        "{"
        <filter info case> # filterCaseFlatImageNoSelection
        <filter info case> # filterCaseFlatImageWithSelection
        <filter info case> # filterCaseFloatingSelection
        <filter info case> # filterCaseEditableTransparencyNoSelection
        <filter info case> # filterCaseEditableTransparencyWithSelection
        <filter info case> # filterCaseProtectedTransparencyNoSelection
        <filter info case> # filterCaseProtectedTransparencyWithSelection
        "}"
    "}"

<filter info case> :=
    <input matting> "," <output matting> ","
    <layer mask flag> "," <blank data flag> "," <copy source flag>

<input matting> :=
    "inCantFilter" |
    "inStraightData" |
    "inBlackMat" |
    "inGrayMat" |
    "inWhiteMat" |
    "inDefringe" |
    "inBlackZap" |
    "inGrayZap" |
    "inWhiteZap" |
```

```
    "inBackgroundZap" |
    "inForegroundZap"

<ouput matting> :=
    "outCantFilter" |
    "outStraightData" |
    "outBlackMat" |
    "outGrayMat" |
    "outWhiteMat" |
    "outFillMask"

<layer mask flag> := "doesNotFilterLayerMasks" | "filtersLayerMasks"
<blank data flag> := "doesNotWorkWithBlankData" | "worksWithBlankData"
<copy source flag> := "copySourceToDestination" |
    "doNotCopySourceToDestination"

<type creator pair> :=
    <OSType> "," <OSType>

<format file type property> :=
    "{"
    <type creator pair>
    "}"

<type creator pair list tail> :=
    <epsilon> |
    "," "{" <type creator pair> "}" <type creator pair list tail>

<type creator pair list> :=
    <epsilon> |
    "{" <type creator pair> "}" <type creator pair list tail>

<read types property> :=
    "{"
    <type creator pair list>
    "}"

<write types property> :=
    "{"
    <type creator pair list>
    "}"

<filtered types property> :=
    "{"
    <type creator pair list>
    "}"

<ostype list tail> :=
    <epsilon> |
    "," "{" <OSType> "}" <ostype list tail>

<ostype list> :=
    <epsilon> |
    "{" <OSType> "}" <ostype list tail>

<read extensions property> :=
    "{"
    <ostype list>
    "}"

<write extensions property> :=
    "{"
    <ostype list>
```

```
    "}"

<filtered extensions property> :=
    "{"
    <ostype list>
    "}"

<saves image resources flag> :=
    "fmtDoesNotSaveImageResources" | "fmtSavesImageResources"

<can read flag> :=
    "fmtCannotRead" | "fmtCanRead"

<can write flag> :=
    "fmtCannotWrite" | "fmtCanWrite"

<write if read flag> :=
    "fmtWritesAll" | "fmtCanWriteIfRead"

<format flags property> :=
    "{"
    <saves image resources flag> ","
    <can read flag> ","
    <can write flag> ","
    <write if read flag>
    "}"

<format maximum size property> :=
    "{"
    <int16>, <int16>
    "}"

<format maximum channels property> :=

<parsable types property> :=
    "{"
    <type creator pair list>
    "}"

<parsable extensions property> :=
    "{"
    <ostype list>
    "}"

<filtered parsable types property> :=
    "{"
    <type creator pair list>
    "}"

<filtered parsable extensions property> :=
    "{"
    <ostype list>
    "}"

<parsable clipboard types property> :=
    "{"
    <ostype list>
    "}"


<animatable filter description> :=
    "{"
    <animatable filter information>, <animatable filter parameters>
```

```
    "}"

<animatable filter information> :=
    {
    <long>, <long>, <long>,
    <animatable filter flag>,
    <long>, <char[32]>, <long>
    }

<animatable filter flag> :=
    "ANIM_FF_HAS_RANDOMNESS" |
    "ANIM_FF_NON_GEOMETRIC" |
    "ANIM_FF_FG_ANIMATABLE" |
    "ANIM_FF_BG_ANIMATABLE" |
    "ANIM_FF_PARAMS_IN_GLOBALS" |
    "ANIM_FF_DIALOG_IN_RENDER" |
    "ANIM_FF_PARAMS_ARE_MAC_HANDLE" |
    "ANIM_FF_PARAMS_ARE_PTR" |
    "ANIM_FF_DOESNT_NEED_DLOG" |
    "ANIM_FF_DONT_DRIVE_ME" |
    "ANIM_FF_RESERVED0" |
    "ANIM_FF_RESERVED1"

<animatable filter parameters> :=
    {
    <char[32]>, <long>,
    <animatable data type>, <animatable ui type>,
    <float64>, <float64>, <float64>, <float64>,
    <animatable parameter flags>, <long>, <long>
    }

<animatable data type> :=
    "ANIM_DT_OPAQUE" |
    "ANIM_DT_CHAR" |
    "ANIM_DT_SHORT" |
    "ANIM_DT_LONG" |
    "ANIM_DT_UNSIGNED_CHAR" |
    "ANIM_DT_UNSIGNED_SHORT" |
    "ANIM_DT_UNSIGNED_LONG" |
    "ANIM_DT_FIXED" |
    "ANIM_DT_UNSIGNED_FIXED" |
    "ANIM_DT_EXTENDED_96" |
    "ANIM_DT_DOUBLE_64" |
    "ANIM_DT_FLOAT_32"

<animatable ui type> :=
    "ANIM_UI_NO_UI" |
    "ANIM_UI_ANGLE" |
    "ANIM_UI_SLIDER" |
    "ANIM_UI_POINT" |
    "ANIM_UI_RECT" |
    "ANIM_UI_COLOR_RGB" |
    "ANIM_UI_COLOR_CMYK" |
    "ANIM_UI_COLOR_LAB"

<animatable parameter flags> :=
    "ANIM_PF_IS_RES_DEPENDENT" |
    "ANIM_PF_SPACE_IS_RELATIVE" |
    "ANIM_PF_RESTRICT_BOUNDS"

<effects pipl version> := {
    <long>, <long>
    }
```

```
<pf spec version> := {
    <long>, <long>
    }

<effect version> := {
    <long>
    }

<effect info flags> := {
    <int16>
    }

<effect global outflags> :=
    "PF_OutFlag_KEEP_RESOURCE_OPEN" |
    "PF_OutFlag_WIDE_TIME_INPUT" |
    "PF_OutFlag_NON_PARAM_VARY" |
    "PF_OutFlag_SEND_PARAMS_UPDATE" |
    "PF_OutFlag_SEQUENCE_DATA_NEEDS_FLATTENING" |
    "PF_OutFlag_I_DO_DIALOG" |
    "PF_OutFlag_USE_OUTPUT_EXTENT" |
    "PF_OutFlag_SEND_DO_DIALOG" |
    "PF_OutFlag_DISPLAY_ERROR_MESSAGE" |
    "PF_OutFlag_I_EXPAND_BUFFER" |
    "PF_OutFlag_PIX_INDEPENDENT" |
    "PF_OutFlag_I_WRITE_INPUT_BUFFER" |
    "PF_OutFlag_I_SHRINK_BUFFER" |
    "PF_OutFlag_WORKS_IN_PLACE" |
    "PF_OutFlag_SQUARE_PIX_ONLY" |
    "PF_OutFlag_CUSTOM_UI" |
    "PF_OutFlag_CUSTOM_NTRP" |
    "PF_OutFlag_REFRESH_UI" |
    "PF_OutFlag_NOP_RENDER" |
    "PF_OutFlag_I_USE_SHUTTER_ANGLE" |
    "PF_OutFlag_I_USE_AUDIO" |
    "PF_OutFlag_I_AM_OBSOLETE" |
    "PF_OutFlag_RESERVED1" |
    "PF_OutFlag_RESERVED2" |
    "PF_OutFlag_RESERVED3" |
    "PF_OutFlag_RESERVED4"

<effect match name> := {
    <string>
    }

<imageformat extension info> :=
    "Input" | "Output" | "File" | "Still" | "Video" | "Framestore" |
    "InteractGet" | "InteractPut" | "InteractPutRevert" |
    "AddFrameNonSeq" | "NoOutputDialog"
```

# 5. Adobe Illustrator

This chapter describes properties and useful resources of Adobe Illustrator for creating plug-ins that work under multiple applications.

**Table 5-1: Adobe Illustrator version and signature information**

| Description | Value |
|---|---|
| Mac OS version | 6.0 |
| Mac OS release date | 2/6/96 |
| Windows version | 4.1 |
| Windows release date | 6/5/95 |
| Backward-compatibility targets Mac, Win | 5.5, 4.1 |
| Signature | 'ART5' |

# Adobe Illustrator and Adobe Photoshop

**Table 5-2: Adobe Illustrator emulating Photoshop host**

| Description | Value |
|---|---|
| Signature | 'ART5' |
| Host version support | 3.0.4 subset, N/A |
| Required adaptor | Photoshop Adapter plug-in |
| Resource | 'PiPL' |
| Supported module types | Filter, Format |
| Plug-in folder default | Adobe Illustrator/Plug-ins/(Gallery Effects) |
| Plug-in aliases | Automatically resolved by Illustrator. |
| Plug-in load order | Loads references, but not code until execution request. |
| *How to access the different plug-ins while using Adobe Illustrator.* | |
| Filter modules | **Filter**»*(sub-menu)* |
| Format modules | **File**»**Save as...** |

### Host version support

Adobe Illustrator and the Photoshop Adapter plug-in emulates the Photoshop 3.0.4 Plug-in API. All of the 3.0.4 API calls and functions are implemented, except:

1. Any callbacks related to Acquire or Export modules.
2. The Color Services suite returns -1 error.
3. HostGetProperty, propInterpolationMethod returns 1 + noErr.
4. HostSetProperty returns -1 error.
5. ProcessEvent does nothing.
6. AdvanceState is not supported in Format writing.
7. Photoshop files are flattened when imported.
8. Multiple channel information is not available.
9. Indexed color mode.
10. Alternate layouts, layers, padding, transparency, and tiling.

# 6. Adobe Illustrator PiPLs

## The Plug-in Propery List Resource

Plug-in Property List, 'PiPL', resources provide a host application information about a plug-in. This information includes indicators about the types and locations of available code, versions, and other dependencies of the plug-in. PiPLs were first used in Adobe Photoshop 3.0 plug-ins. They have been adapted for use with the Adobe Illustrator 6.0 API by ignoring certain Photoshop specific properties and defining others. The general PiPL definition is the same as that for Photoshop and these notes are based on a description from the Photoshop 3.0 SDK. This chapter describes what you need to get quickly started with PiPLs in Illustrator.

### PIPropertyList

```
typedef struct PIPropertyList
{
    int32 version;
    int32 count;
    AIProperty properties[1];
} PIPropertyList;
```

**Table 6-1: PIPropertyList structure**

| Type | Field | Description |
|---|---|---|
| int32 | version | Current version is 0. |
| int32 | count | Number of properties in the 'PiPL'. 0=no properties. |
| AIProperty | properties | A variable length array of property data structures. |

### Properties

```
typedef struct AIProperty
{
    OSType vendorID;
    OSType propertyKey;
    int32  propertyID;
    int32  propertyLength;
    char   propertyData [1];
    /* Implicitly aligned to multiple of 4 bytes. */
} AIProperty;
```

**Table 6-2: AIProperty structure**

| Type | Field | Description |
|---|---|---|
| OSType | vendorID | This field identifies the vendor defining this property type. This allows other vendors to define their own properties in a way that does not conflict with either Adobe or other vendors. It is recommended that a registered application creator code be used for the vendorID to ensure uniqueness. All Photoshop properties use the vendorID '8BIM'. All Illustrator 6.0 plug-ins use the vendorID 'ADBE'. |
| OSType | propertyKey | Property type, detailed in table 6-3. |

**Table 6-2: AIProperty structure (Continued)**

| Type | Field | Description |
|---|---|---|
| int32 | propertyID | =0. Used to store more than one property of a given type. Reserved for future use. |
| int32 | propertyLength | Length of `propertyData`. Does not include any padding bytes to achieve four byte alignment. May be zero. |
| char | propertyData | Variable length field containing contents of this property. Any values may be contained. |

Each property must be padded such that the next property begins on a four byte boundary.

## Adobe Illustrator properties in the Mac OS and Windows

Specific properties can be extended in an upwardly compatible fashion by adding extra data at their end. The length field will allow an application to determine how much data is present, so optional properties can be omitted without concern. This is different from a fixed length structure where omitted fields must be given a default value.

It is intended for `PiPLs` to collect all plug-in metadata in a single place. this is useful for cross-platform development, since Windows lacks a resource management mechanism.

The '`PiPL`' format is fairly portable in that everything is four byte aligned. All `OSType` and `int32` fields are represented in native byte order for a given platform so bytes of informationally indential `PiPLs` will differ between big-endian machines that run the Mac OS, and little-endian machines running Windows. The bytes of the `PiPL` section of a Windows binary resource will be indentical, but reversed, to the same resource in the Mac OS. This should not be of too much concern. As long as you use the pre-defined plug-in data types (table 6-3), they will be interpreted and stored correctly.

**Note:** An undefined `OSType` will not be converted automatically. It is normally interpreted as a `long` and you must supply the chars in reverse order for Windows implementation.

The Illustrator API byte order is always big-endian.

## Adobe Illustrator basic data types

The following types are used to define properties:

**Table 6-3: Adobe Illustrator basic data types**

| Name | Description |
|---|---|
| int16, int32 | 16 and 32 bit integers. Stored in native byte order. |
| OSType | Same as `int32`. Typically denotes Mac OS 4 character filetypes like '`PiPL`'. |
| PString | Pascal style string where byte 1=length and content bytes follow. |
| CString | C style string where the content bytes are terminated by `NULL`. |
| Structures | Represented as would be in memory on the target platform. Native padding and alignment constraints are observed. |
| Arrays | Represented as a contiguous set of entries in the '`PiPL`' with native padding and alignment constraints observed. |

## General properties

The following general property keys are recognized by Adobe Illustrator.

**Table 6-4: Adobe Illustrator general property keys**

| Type | Name | Key | Description |
|---|---|---|---|
| OStype | AIKindProperty | 0x6b696e64L ('kind') | Type or kind of plug-in. Adobe Illustrator = 'ARPI'. Photoshop filter='8BFM'. |
| int32 | AIVersionProperty | 0x69767273L ('ivrs') | Version of the plug-in interface expected by the plug-in. A version change should be assumed to be 100% incompatible with other versions. |

## Code Descriptor Properties

Code descriptors tell the host application the location of a plug-in's code. More than one code descriptor may be included to build a "fat" plug-in which will run on different types of machines. Illustrator does not support emulated plug-ins, so if a code descriptor for the running platform does not exist, the plug-in will not be loaded. Illustrator exports function suites with appropriate functions for the type of code that is loaded. For PowerPC code this means native function pointers will be provided. Routine descriptor operations are not required either in calling the plug-in nor for the plug-in to invoke Illustrator callback functions.

**Table 6-5: Adobe Illustrator code descriptor properties**

| Type | Name | Key | Description |
|---|---|---|---|
| AI68kCodeDesc | AI68KCodeProperty | 0x6d36386bL ('m68k') | This descriptor indicates a 68K code resource. The type for this property is:<br><br>`typedef struct AI68KCodeDesc {`<br>`  OSType resourceType;`<br>`  int16 resourceID;`<br>`} AI68KCodeDesc;`<br><br>Any resource type may be used, but the convention for Illustrator 6.0 plug-ins is 'ARPI', number 16000 |
| AICFMCodeDesc | AIPowerPCCodeProperty | 0x70777063L ('pwpc') | This descriptor indicates a PowerPC code fragment in the data fork of the plug-in file. The type for this property is as follows:<br><br>`typedef struct AICFMCodeDesc {`<br>`  long fContainerOffset;`<br>`  long fContainerLength;`<br>`  char fEntryName[1];`<br>`} AICFMCodeDesc;`<br><br>Described in table 6-6. |

**Table 6-6: AIPowerPCCodeProperty properties**

| Type | Field | Description |
|---|---|---|
| long | fContainerOffset | Data fork offset to the code fragment start. This allows more than one code fragment based plug-in per file. |
| long | fContainerLength | Length of the code fragment. If the fragment extends to the end of the file or is the only fragment, the container length may be 0. |
| char | fEntryName | Not currently implemented in Adobe Illustrator 6.0.<br><br>Pascal string used to lookup the address of the function to call within the fragment. In order for the Code Fragment Manager to find an entrypoint by name, that name must be an exported symbol of the code fragment. If NULL, the default entrypoint will be used. fEntryName allows a single code fragment to contain more than one plug-in. |

# Import and Export Properties

Import and export properties are used by plug-ins to declare their interdependencies with other plug-ins. Plug-ins may depend on the existence of another plug-in even if it doesn't explicitly export callback functions.

For instance, one vendor's plug-ins may expand upon the functionality of another's. This dependency can be expressed by declaring (exporting) a plug-in's existence. Such exported properties could include menu items, tools, or filters.

## Importing

Plug-ins access callback functions by first importing function suites. The plug-in declares the suites and other functionality on which it depends using an *import property list*.

## Exporting

Plug-ins can extend the functionality of the API by exporting new function suites. A plug-in must have at least one export property, which is often the name of the plug-in.

The loading order of plug-ins becomes important when one plug-in depends on a suite exported by another, as the exporting plug-in must be loaded first to initialize function lists and other values. To ensure that the interdepencies of plug-ins are handled correctly, plug-ins declare in advance the function suites they import and export. Illustrator will use this information when loading and executing plug-ins. The suite import and export information is declared in the PiPL resource.

**Table 6-7: Adobe Illustrator import and export properties**

| Type | Name | Key | Description |
|---|---|---|---|
| AIImportsList | AIImportList | 0x696D7074L ('impt') | This descriptor contains the list of dependencies that a plug-in imports or exports:<br><br>`typedef struct AIImportExportDesc`<br>`{`<br>`  long fCount;`<br>`  AIIEListsDesc fImpExp[1];`<br>`} AIImportExportDesc;`<br><br>See table 6-8 and 6-9. |
| AIExportsList | AIExportList | 0x65787074L ('expt') | |

**Table 6-8: AIImportExportDesc properties**

| Type | Field | Description |
|---|---|---|
| long | fCount | The number of suites imported by the plug-in. |
| long | fImpExp | Variable length list describing the suites needed by the plug-in. |

### AIIEListDesc

```
typedef struct AIIEListDesc
{
    long    fLength;
    CString fName;   // padded to four bytes
    long    fVersion;
} AIIEListDesc;
```

**Table 6-9: AIIEListDesc properties**

| Type | Field | Description |
|---|---|---|
| long | fLength | The total length (including 4 bytes for this field) of the `AIIEListDesc` record. |
| CString | fName | C-style string with the name of the suite to be imported or exported. The usable names of suites are found in the API documentation and header files. |
| long | fVersion | The version of a suite to use. Supported versions are listed in the API documentation. |

## Dynamically Declared Properties

Illustrator also provides a mechanism for declaring properties in a more dynamic fashion. If the 'impt' or 'expt' property, or both, do not exist, the plug-in will be sent two messages requesting the properties:

```
kAISelectorAcquireProperty
kAISelectorReleaseProperty
```

Your plug-in should build the appropriate property information and return a pointer to that information. Determine the message type in main():

```
FXErr main( char *selector, void *stuff )
{
    if ( strcmp( selector, kSelectorAIAcquireProperty ) == 0 )
        error = AcquireProperty( stuff );
    else if (strcmp(selector, kSelectorAIReleaseProperty) == 0 )
        error = ReleaseProperty( stuff );
    else
        // process any other messages
}
```

Next, call a routine to create or release the property structure. When creating the PiPL in memory, use platform memory allocation routines or declare the property internally as static data.

The data passed with these messages is:

```
typedef struct
{
      AIPluginData d;
      OSType vendorID;        // same as PiPL definition
      OSType propertyKey;     // same as PiPL definition
      long propertyID;        // as always, 0
      void *property;         // return the property here
      long refCon;            // for plug-in's use. Set on acquire,
                              //    given back on release
      Boolean cacheable;      // most likely true
} AIPropertyMessage;
```

When the kAISelectorAcquireProperty message is received, the vendorID and propertyKey fields define the requested property, either 'impt' or 'expt'. Field propertyID=0, as defined. Based on the request, the plug-in must create the property in memory exactly as defined in the PiPL description and return a pointer to this memory block in the property field. If the information in the property data will not change, set cacheable=TRUE. Cacheable properties may be stored by the host in a startup preferences file.

When the kAISelectorReleaseProperty message is received, the plug-in should free the memory allocated to create the property.

# Adobe Illustrator SDK information and samples

## Working with PiPLs

The basic PiPL resource any fat plug-in will need is found in the file "Basic PiPL.rsrc" in the "PiPL example" folder of the sample code on the Adobe Illustrator SDK. You can add Import and Export property lists to this resource or specify them dynamically using the method shown in most of the sample code.

If you need to edit a PiPL resource, you will need to use a resource compiler or the program *Resorcerer* by Mathemæsthetics, Inc. There is a Resorcerer template in the file Basic PiPL.rsrc with the Adobe Illustrator SDK that will make editing property list straightforward. ResEdit resource templates cannot handle a resource as complex as a PiPL.

## Sample PiPLs

The sample code on the Adobe Illustrator SDK provides examples of both methods of using PiPLs. The folder "shell w/ PiPL resource" has a resource based PiPL. The routines are provided in the sample code to create the property data from a modifiable structure.

# 7. Adobe PageMaker

This chapter describes properties and useful resources of Adobe PageMaker for creating plug-ins that work under multiple applications.

**Table 7-1: Adobe PageMaker version and signature information**

| Description | Value |
|---|---|
| Mac OS version | 6.0 |
| Mac OS release date | 6/1/95 |
| Windows version | 6.0 |
| Windows release date | 8/1/95 |
| Backward-compatibility targets Mac, Win | 5.0, 5.0 |
| Signature | 'ALD6' |

# Adobe PageMaker and Adobe Photoshop

**Table 7-2: Adobe PageMaker emulating Adobe Photoshop host**

| Description | Value |
|---|---|
| Signature | '8BIM' |
| Host version support | 3.0.5, 3.0.5 |
| Required adaptor | Photoshop Effects extension |
| Resource | 'PiPL' |
| Supported module types | Filter |
| Plug-in folder default | Adobe PageMaker 6.0/RSRC/Plugins/Effects |
| Plug-in aliases | Automatically resolved by PageMaker. |
| Plug-in load order | Loads references, but not code until execution request. |
| *How to access the different plug-ins while using Adobe PageMaker:* | |
| Filter modules | **Element**»**Image**»"Photoshop Effects..." |

### Host version support

Adobe PageMaker and the Photoshop Effects extension emulates the Photoshop 3.0.5 Plug-in API. All of the 3.0.5 API calls and functions are implemented, except Adobe PageMaker:

1. executes only 680x0 plug-in code for, and does not support any call-backs related to Acquire, Export, or File Format modules;
2. requires the Microsoft OLE extension;
3. does not support the Color Services callback.

### Nomenclature

An Adobe PageMaker "Filter" is the same as Adobe Photoshop's "File Format." Adobe PageMaker's "Plugin" or "Effect" is the same as Adobe Photoshop's "Filter".

Before version 6.0, Adobe PageMaker used the term "Addition" for what is now called a "Plug-in".

### Plug-in execution

Plug-in execution is vastly different in Adobe PageMaker with the Photoshop Effects extension then any other implementation. Photoshop Filter plug-ins are executed in this order:

1. User selects "Photoshop Effects..."
2. Dialog appears with selected image's name ("Flower.tif"), Name of new image to create ("Flower1.tif"), and name of filter to use.
3. User selects file names and filter from pop-up menu. Clicks "OK".
4. Original image is loaded.
5. Plug-in is called with entire image data.
6. Resulting filtered image is saved.
7. Resulting image is placed with same position and parameters as original image, replacing all original references.

**Note:** Because PageMaker generates a new image every time a filter is executed, it is important to give the user as much preview feedback as possible while they are modifying the plug-in parameters. If you don't have a preview window in your plug-in dialog, you might want to implement it to support PageMaker users.

# 8. Adobe PhotoDeluxe

This chapter describes properties and useful resources of Adobe PhotoDeluxe for creating plug-ins that work under multiple applications.

**Table 8-1: Adobe PhotoDeluxe version and signature information**

| Description | Value |
|---|---|
| Mac OS version | 1.0 |
| Mac OS release date | 1/1/96 |
| Windows version | None |
| Windows release date | N/A |
| Backward-compatibility targets Mac, Win | 3.0, N/A |
| Signature | 'PHUT' |

# Adobe PhotoDeluxe and Adobe Photoshop

**Table 8-2: Adobe PhotoDeluxe emulating Adobe Photoshop host**

| Description | Value |
|---|---|
| Signature | '8BIM' |
| Host version support | 3.0.5 LE, 3.0.5 LE |
| Required adaptor | N/A |
| Resource | 'PiPL' |
| Supported module types | Acquire, Export, Filter, Format |
| Plug-in folder default | Adobe PhotoDeluxe/Plug-ins |
| Plug-in aliases | Automatically resolved by PhotoDeluxe. |
| Plug-in load order | Loads references, but not code until execution request. |
| *How to access the different plug-ins while using Adobe PhotoDeluxe:* | |
| Acquire modules | **File»Acquire** |
| Export modules | **File»Export** |
| Filter modules | **File»Long menus**; click “On your own”; **Effects**.<br>At this time there is no API to add to the “Cool” or other tabbed menus. |
| Format modules | **File»Export**»“File formats...” |

### Host version support

Adobe PhotoDeluxe emulates the Photoshop 3.0.5 LE Plug-in API. All of the 3.0.5 API calls and functions are implemented, except Adobe PhotoDeluxe only executes the 680x0 code of the plug-in, and does not support:

1. CMYK and Lab modes in `PSDisplayPixels` and `PSSampleImage`
2. CMYK, Lab, and XYZ ColorServices callbacks such as `RGBtoCMYK` and `CMYKtoRGB`
3. The `GetPathName()` callback
4. The path properties `numberOfPaths`, `pathName`, `pathContents`, `targetPathIndex`, `workPathIndex`, `clippingPathIndex`, and `caption`
5. AGM and AGM memory host callbacks
6. `GetDuotoneInfo` and `SetDuotoneInfo`.

# 9. Adobe Photoshop

This chapter describes properties and useful resources of Adobe Photoshop for creating plug-ins that work under multiple applications.

**Table 9-1: Adobe Photoshop version and signature information**

| Description | Value |
|---|---|
| Mac OS version | 4.0 |
| Mac OS release date | 11/18/96 |
| Windows version | 4.0 |
| Windows release date | 11/18/96 |
| Backward-compatibility targets Mac, Win | 2.5, 2.5 |
| Signature | '8BIM' |

## Host emulators

Table 9-2 is a list of known hosts that emulate the Adobe Photoshop plug-in API. Refer to the individual chapters and SDKs of each application for Implementation issues and emulation caveats.

**Table 9-2: Known host emulators and supported versions**

| Host | Versions supported (Mac, Win) | Modules supported |
|---|---|---|
| Adobe After Effects | 3.0, 3.0 | Filter, Format, Parser |
| Adobe Illustrator | 3.0.4 subset, N/A | Filter, Format |
| Adobe PageMaker | 3.0.5, 3.0.5 | Filter |
| Adobe PhotoDeluxe | 3.0.5 LE, 3.0.5 LE | Acquire, Export, Filter, Format |
| Adobe Premiere | 2.5, 3.0 | Filter |

# 10. Adobe Photoshop PiMIs

PiMI (pronounced “pimmy”) resources have been superceded by PiPL resources, but you may need to include a PiMI resource if your want your plug–in module to work with older (pre–3.0) versions of Adobe Photoshop. Adobe recommends that you also create a PiPL resource for your plug–in, as this will give you greater control over its operation under 3.0.

If your plug–in module is designed to be used only with Photoshop 3.0 or later, we recommend, for backward-compatibility, you create a PiMI resource, and provide alternate code for the suites that are unavailable in version 2.5.

Older PiMI based plug–in modules are still fully supported in Photoshop 3.0. This is accomplished by converting the 'PiMI' resource into a 'PiML' resource when the plug–in is first scanned. Since PiMLs are cached in Photoshop’s preferences file, this conversion only happens once.

If you want your plug–in to work with versions of Photoshop prior to 3.0, you must create a 'PiMI' resource.

A PiMI resource is a fixed format structure which originally contained only a version number. With the evolution of Photoshop’s plug–in interface, this structure expanded to include other information. The addition of multiple plug–in types resulted in the PiMI becoming a variant record with generic data at the beginning and a type specific data at the end. Further plug–in interface evolution required more complex metadata, such as an array of allowable file types for file format plug–ins.The combination of variant and variable sized fields in the 'PiMI' made writing resource templates for them very difficult. Requirements for new plug–in metadata in Photoshop 3.0 introduced further complexities. The more general and flexible 'PiML' mechanism was designed to address these issues.

The PiMI resource consists of two pieces: general information applicable to all (or most) plug–in types followed by type specific info. Since the information proceeds serially, however, all fields must be filled in through and including the last field supplied. Your plug–in should either just include the version number information, or it should include all of the information documented here.

```
typedef struct PlugInInfo
{
    short               version;
    short               subVersion
    short               priority;
    short               generalInfoSize;
    short               typeInfoSize;
    short               supportsMode;
    OSType              requireHost;
} PlugInInfo;
```

**Table 10-1: Adobe Photoshop PlugInInfo (PiMI) structure**

| Type | Field | Description |
|---|---|---|
| short | version | Major version number for the interface used by the plug–in. Required. |
| short | subVersion | Minor version number for the interface used by the plug–in. Required. |
| short | priority | The priority of this plug–in when it loads. This is only used for extension modules. |
| short | generalInfoSize | The size of the general plug–in information. |
| short | typeInfoSize | The size of the type–specific plug–in information. This information follows `requiredHost`. See the SDK for type specifics. |
| short | supportsMode | A bitmap describing the image modes supported by the plug–in. This field applies to Export, Filter, and File Format plug–ins. If not present, Photoshop assumes all image modes. This field is one of the ways Photoshop decides whether to dim plug–ins in menus.<br><br>Since not all plug–in hosts may respect this field, your plug–in module should still check that it can handle the image mode it has been requested to process. The bits in the bitmap correspond to the `plugInMode` constants in `PIGeneral.h` (i.e. bit 0 corresponds to bitmaps, bit 1 to grayscale, etc.). |
| short | requireHost | If your plug–in requires a particular plug–in host, you should specify the signature for that host here. If you do not require a particular plug–in host, you should fill this field with spaces.<br><br>Photoshop will not load plug–in modules which require a plug–in host other than Photoshop's `'8BIM'` signature. You should not count on other applications that support the Photoshop plug–in architecture to behave in a similar fashion. |

# 11. Adobe Photoshop PiPLs

A Plug–In Property List, called a 'PiPL' (pronounced “pipple”) is a flexible, extensible data structure for representing a plug–in module’s metadata.

PiPLs contain all the information Photoshop needs to identify and load plug–in modules, as well as flags and other static properties that control the operation of each plug–in. Your plug–in module should contain one or more 'PiPL' structures.

Plug–in Property Lists were introduced with version 3.0 of Adobe Photoshop. They replace the older Plug–in Module Information structure, or 'PiMI'. PiMI resources were used with versions of Photoshop prior to 3.0, and are discussed in more detail in the previous chapter.

## Property structures and property lists

Plug–in *property structures* (or *properties*) are the basic units of information stored in a *property list*. Properties are variable length data structures, which are uniquely identified by a vendor code, property key, and ID number. The valid properties and formal grammar are documented later in this chapter.

## Creating PiPL resources

Under the Mac OS, PiPLs are stored as Macintosh resources. Under Windows, PiPLs are stored as Windows resources.

On the Macintosh, you can create and edit PiPL resources with a text editor and the *Rez* compiler, or you can use a graphical resource editor like *Resorcerer*. ResEdit cannot edit PiPL resources. If you are unfamiliar with the format of Rez source code, refer to the appropriate Apple documentation. The Photoshop SDK includes a Macintosh Rez file, PIGeneral.r, which provides a complete definition of the PiPL property types.

The Windows version of the Photoshop SDK also includes a “PiPL Parser” utility, CNVTPIPL.EXE, to transform a Macintosh “.r” source file into a Windows “.rc” resource file.

If you are developing for both the Macintosh and Windows platforms, you can easily convert your Macintosh PiPL resources into Windows’ custom PiPL format using CNVTPIPL.EXE. This enables you to keep just one copy of your PiPL information, and saves you the headache of converting PiPLs by hand.

Even if you are developing a plug–in module only for Windows, you are strongly encouraged to use the Macintosh Rez language to create the PiPLs, and then use CNVTPIPL.EXE to convert them. It is much easier to create the PiPLs this way since CNVTPIPL.EXE handles padding and byte–ordering issues for you automatically. If you use a Windows resource editor, you will have to remember to byte–swap fields where necessary.

### Creating or modifying PiPL resources in Windows

When the Macintosh development environment is not available for creating the PiPL resource, or you only want to make a minor change while still in your Windows development environment, follow these steps:

1. Open the resource rez file for the plug-in, *plugInName*.r (such as dissolve.r)
2. Make any desired PiPL changes to the resource file.
3. Save the file.
4. Rebuild the project.

The makefile builds the resources for your plug-in in this order:

1. PiPL.TMP is generated by feeding *plugInName*.r through the C Pre-Processor.
2. PiPL.RSC is then generated by feeding PiPL.TMP through CNVT-PIPL.EXE.
3. *plugInName*.rc is created, which includes PiPL.RSC.

**Note:** Editing PiPL.RSC, PiPL.TMP or *plugInName*.rc only modifies your PiPL temporarily, and will not create a valid PiPL. Your plug-in will compile, but Photoshop will not recognize it. Only edit your PiPL resource via the *plugInName*.r file.

## Loading PiPL resources

When Photoshop launches, it scans all plug–in files for 'PiPL' resources. Historically, each type of plug–in had its own file type.

File types are only a matter of convention for 'PiPL' based plug–in modules. All known plug-in file types are searched for 'PiPL' resources and for those that are found, the information contained therein is used to determine the type of plug–in, code location, etc.

If no 'PiPL' resources are found in a plug–in file, the 'PiMI' search algorithm is used. This allows you to place both 'PiPL' and 'PiMI' resources in a plug–in module to make it compatible with both version 2.5 and 3.0.x.

## Plug–in property lists

The plug–in property list structure has a version number and a count followed by one or more property structures.

```
typedef struct PIPropertyList
{
    int32               version;
    int32               count;
    PIProperty          properties[1];
} PIPropertyList;
```

**Table 11-1: PIPropertyList structure**

| Type | Field | Description |
|---|---|---|
| int32 | version | Current version is 0. |
| int32 | count | Number of properties in the 'PiPL'. 0=no properties. |
| PIProperty | properties | A variable length array of property data structures. |

## Plug–in properties

Each property has a vendor code, a key, an ID, a length field.

```
typedef struct PIProperty
{
```

```
    OSType              vendorID;
    OSType              propertyKey;
    int32               propertyID;
    int32               propertyLength;
    char                propertyData [1];
    /* Implicitly aligned to multiple of 4 bytes. */
} PIProperty;
```

**Table 11-2: PIProperty structure**

| Type | Field | Description |
|---|---|---|
| OSType | vendorID | The vendor defining this property type. This allows other vendors to define their own properties in a way that does not conflict with either Adobe or other vendors. It is recommended that a registered application creator code be used for the `vendorID` to ensure uniqueness. All Photoshop properties use the vendorID '8BIM'. |
| OSType | propertyKey | Property type, detailed in table 11-3. |
| int32 | propertyID | =0. Used to store more than one property of a given type. Reserved for future use. |
| int32 | propertyLength | Length of `propertyData`. Does not include any padding bytes to achieve four byte alignment. May be zero. |
| variable | propertyData | Variable length field containing contents of this property. Any values may be contained. |

## General properties

These properties are common to all types of plug–in modules. The names of the properties (such as “PIKindProperty”) are the same as the #define names for the corresponding property keys.

**Table 11-3: Adobe Photoshop general property keys**

| Type | Name | Key | Description |
|---|---|---|---|
| OStype | PIKindProperty | 0x6b696e64L ('kind') | Type or kind of plug-in.<br>'ARPI'=Adobe Illustrator<br>'8BXM'=Accelerator extension<br>'8BAM'=Import module<br>'8BEM'=Export module<br>'8BFM'=Filter module<br>'8BIF'=Format module<br>'8BSM'=Selection module<br>'8BYM'=Parser module |
| int32 | PIVersionProperty | 0x76657273L ('vers') | Major and minor version number indicating which revision of the plug–in interface this plug–in was written for. The major version number indicates incompatible changes while the minor version number indicates incremental enhancements. The major version number is encoded in the most significant 16 bits of the 32 bit version number, the minor version number is encoded in the least significant 16 bits.<br>There are separate version numbers for each kind of plug–in. The current version for a given kind of plug–in is defined by a preprocessor macro in the header file defining the interface for that plug–in type. |
| int16 | PIPriorityProperty | 0x70727479L ('prty') | Plug-in load order. Also used to control the order in which items with the same name show up in menus.<br>Lower numbers (including negative ones) load first. If NULL, the default is zero. |
| FlagSet | PIImageModesProperty | 0x6d6f6465L ('mode') | Which image modes the plug–in supports. Adobe Photoshop, has 11 modes: bitmap, grayscale, indexed, RGB, CMYK, HSL, HSB, multi–channel, duotone, Lab, gray 16, and RGB 48.<br>This property determines whether your plug–in will be active (black) or inactive (gray) in Photoshop’s menus based on the current document’s image mode. |
| CString | EnableInfo | 0x656E626CL ('enbl') | String of modula-like expressions that determine whether plug-in is enabled in menu. See below. |
| OSType | PIRequiredHostProperty | 0x686f7374L ('host') | Creator code of required host, such as '8BIM' for Adobe Photoshop. |
| PString | PICategoryProperty | 0x63617467L ('catg') | In the **Filter** menu, what sub–menu to list this plug–in. |
| PString | PINameProperty | 0x6e616d65L ('name') | Plug-in menu name for module in PICategoryProperty sub-menu. |

## EnableInfo

EnableInfo is a string of expressions that, upon evaluation, must all be true for the plug-in to be enabled in the menu.

**Table 11-4: EnableInfo grammar**

| Type | Description |
|---|---|
| booleanExpression | conjunction {“||” conjunction} |
| conjunction | relation {“&&” relation} |
| relation | equality {relationOperator equality} |
| equality | simpleExpression {equalityOperator simpleExpression} |
| simpleExpression | term {addOperator term} |
| term | factor {mulOperator factor} |
| factor | integer \| intrinsic \| ident \| “(“ booleanExpression “)” \| “(“ simpleExpression “)” \| “+” factor \| “–” factor \| “!” factor |
| integer | digit {digit} |
| intrinsic | limitFunction \| dimFunction \| namedParameterFunction |
| limitFunction | (“min” \| “max”) “(” simpleExpression “,” simpleExpression {“,” simpleExpression} “)” |
| dimFunction | “dim” “(”simpleExpression “,” simpleExpression “)” |
| ident | (alpha \| “_”) {alpha \| digit \| “_”} |
| mulOperator | “*” \| “/” |
| addOperator | “+” \| “–” |
| equalityOperator | “==” \| “!=” |
| relationOperator | “<” \| “<=” \| “>=” \| “>” |
| inFunction | “in” “(” simpleExpression {“,” simpleExpression} “)” |

**Table 11-5: EnableInfo constants**

| Value | Description |
|---|---|
| true | Boolean true |
| false | Boolean false |
| BitmapMode | Bitmap mode |
| GrayScaleMode | Grayscale mode |
| IndexedColorMode | Indexed color mode |
| RGBMode | RGB color mode |
| CMYKMode | CMYK color mode |
| HSLMode | HSL color mode |
| HSBMode | HSB color mode |
| MultichannelMode | Multichannel mode |
| DuotoneMode | Duotone mode |
| LabMode | Lab color mode |
| Gray16Mode | Grayscale mode, 16 bits per channel |
| RGB48Mode | RGB color mode, 16 bits per channel |

**Table 11-6: EnableInfo variables**

| Variable | Description |
|---|---|
| PSHOP_ImageMode | Image mode |
| PSHOP_ImageDepth | Image depth |
| PSHOP_HasLayerMask | Boolean for presence of layer mask |
| PSHOP_HasSelectionMask | Boolean for presence of selection mask |
| PSHOP_HasTransparencyMask | Boolean for presence of transparency mask |
| PSHOP_NumTargetChannels | Number of target channels |
| PSHOP_NumTrueChannels | Number of image channels |
| PSHOP_IsTargetComposite | Boolean for whether flattened |
| PSHOP_ImageWidth | Width of the image |
| PSHOP_ImageHeight | Height of the image. |

**Table 11-7: Operator precendence**

| Variable | Description |
|---|---|
| \|\| | Or |
| && | And |
| + - | Addition or subtraction |
| * / | Multiple or divide |
| < <= >= > | Less than, less than or equal to, greater than or equal to, greater than |
| == != | Equals, does not equal |
| (..) in() max() min() unary: + - ! | Functions, increment, decrement, not |

**Result of undefined values**

The result of any arithmetic operation where at least one of the operands is undefined, or an undefined variable, results in FALSE. The result of a compare (see `relationOperator`) where at least one of the operands is undefined reults in FALSE.

Boolean values are treated as in C/C++ where non-zero is TRUE and zero is FALSE with the exception that an undefined value is also `false`.

**inFunction**

`inFunction` returns `true` if the first parameter is equal to at least one of the following parameters. A typical use might be to see if the image mode is RGB, CMYK, or Lab:

```
in(  PSHOP_ImageMode, RGBMode, CMYKMode, LabMode )
```

## Code descriptor properties

Code descriptors tell Photoshop the type and location of a plug–in's code. More than one code descriptor may be included to build a "fat" plug–in which will run on different types of machines. Photoshop will select the best performing option. Photoshop makes sure that the callback structure is filled in with appropriate functions for the type of code that is loaded. So for PowerPC code, native function pointers will be provided and routine descriptor operations are not required either in calling the plug–in or for the plug–in to invoke Photoshop callback functions.

**Note:** In Windows, the `CNVTPIPL.EXE` utility only recognizes the "`PIWin32X86CodeProperty`" property. It ignores all Mac–specific properties described in this section.

**Table 11-8: Adobe Photoshop code descriptor properties**

| Type | Name | Key |
|---|---|---|
| PI68kCodeDesc | PI68KCodeProperty | `0x6d36386bL ('m68k')` |
| This descriptor indicates a 68K code resource. The type for this property is:<br><br>`typedef struct PI68KCodeDesc`<br>`{`<br>`  OSType resourceType;`<br>`  int16 resourceID;`<br>`} PI68KCodeDesc;`<br><br>Any resource type may be used, but types of `PIKindProperty` from table 11-3 are strongly recommended. | | |
| PI68kCodeDesc | PI68KFPUCodeProperty | `0x36386670L ('68fp')` |
| This descriptor is just like a `PI68KCodeDesc` except it will only be used on Macintosh machines that are equipped with FPU hardware. This allows vendors to easily ship plug–ins that take advantage of FPU hardware but still run on non–FPU Macs. | | |
| PICFMCodeDesc | PIPowerPCCodeProperty | `0x70777063L ('pwpc')` |
| This descriptor indicates a PowerPC code fragment in the data fork of the plug-in file. The type for this property is as follows:<br><br>`typedef struct PICFMCodeDesc`<br>`{`<br>`  long fContainerOffset;`<br>`  long fContainerLength;`<br>`  char fEntryName[1];`<br>`} PICFMCodeDesc;`<br><br>Described in table 11-9. | | |
| PIWin32X86CodeDesc | PIWin32X86CodeProperty | `0x77783836L ('wx86')` |
| This code descriptor is used for 32 bit Windows DLLs, and contains the DLL's entrypoint name.<br><br>`typedef struct PIWin32X86CodeDesc`<br>`{`<br>`  char  fEntryName[1];`<br>`} PIWin32X86CodeDesc;`<br><br>The `NULL`-terminated string may need to be padded with additional `NULL`s to satisfy the 4–byte alignment requirement. | | |

**Table 11-9: PICFMCodeDesc structure**

| Type | Field | Description |
|---|---|---|
| long | fContainerOffset | Data fork offset to the code fragment start. This allows more than one code fragment based plug-in per file. |
| long | fContainerLength | Length of the code fragment. If the fragment extends to the end of the file or is the only fragment, the container length may be 0. |
| Pstring | fEntryName | Pascal string used to lookup the address of the function to call within the fragment. In order for the Code Fragment Manager to find an entrypoint by name, that name must be an exported symbol of the code fragment. If NULL, the default entrypoint will be used. fEntryName allows a single code fragment to contain more than one plug-in. |

## Color Picker–specific properties

**Table 11-10: Adobe Photoshop color picker-specific properties**

| Type | Name | Key |
|---|---|---|
| pstring | PickerID | 0x6873746DL ('pnme') |
| This property is a unique string (UUID or your own ™/© string) that differentiates one color picker from another. | | |

All color pickers must have a unique ID so that they may be accessed correctly by the host. Menu entries are not sufficient to differentiate color pickers.

**Note:** If a color picker does not have a unique ID, or two loaded color pickers have the same ID, then the first color picker in will always be the one executed, despite what appears in the pop-up menu.

## Export–specific properties

This property is only applicable to Export plug–in modules.

**Table 11-11: Adobe Photoshop export-specific properties**

| Type | Name | Key | Description |
|---|---|---|---|
| FlagSet | PIExpFlagsProperty | `0x65787066L ('expf')` | This property indicates that the plug–in can see transparency data. To indicate this, set `PIExpSupportsTransparency`. |

## Filter–specific properties

These properties are only applicable to Filter plug–in modules.

**Table 11-12: Adobe Photoshop filter-specific properties**

| Type | Name | Key |
|---|---|---|
| 7 * 4-bytes | PIFilterCaseInfoProperty | 0x66696369L ('fici') |
| The key feature of Photoshop 3.0 is support for dynamically composited layers of image data.<br><br>A layer consists of color and transparency information for each pixel it contains. Previous versions of Photoshop did not have a transparency component. Completely transparent pixels have an undefined color. Filters will likely affect transparency data as well as color data. This is especially true for filters which produce spatial distortions.<br><br>Photoshop 3.0 offers flexibility in how transparency data is presented to filters. The filter case info property controls the filtering process and presentation of data to the plug–in. This property provides information to Photoshop about what image data cases the plug–in supports. Photoshop then compares the current filtering situation to the supported cases and chooses the best fitting case. The image data is then presented in that case. If none of the supported cases are usable, the filter will be disabled.<br><br>The case properties are an array of seven four byte entries, one for each case, detailed in table 11-13. | | |

### FilterCaseInfo

Each of the 7 elements of the array contains a 4–byte `FilterCaseInfo`:

```
typedef struct FilterCaseInfo
{
    char inputHandling;
    char outputHandling;
    char flags1;
    char flags2;
} FilterCaseInfo;
```

**Table 11-13: Filter cases**

| Item | Name | Description |
|---|---|---|
| 1 | filterCaseFlatImageNoSelection | This is a background layer or a flat image. There is no transparency data or selection. |
| 2 | filterCaseFlatImageWithSelection | No transparency data, but a selection may be present. The selection will be presented as mask data. |
| 3 | filterCaseFloatingSelection | Image data with an accompanying mask. |
| 4 | filterCaseEditableTransparencyNoSelection | Layer with transparency editing enabled and no selection. |
| 5 | filterCaseEditableTransparencyWithSelection | Layer with transparency editing enabled and a selection. |
| 6 | filterCaseProtectedTransparencyNoSelection | Layer with transparency editing disabled and no selection. |
| 7 | filterCaseProtectedTransparencyWithSelection | Layer with transparency editing disabled and a selection. |

If the editable transparency cases are unsupported, then Photoshop will try the corresponding protected transparency cases. This governs whether the filter will be expected to filter the transparency data with the color data.

If the protected transparency case without a selection is disabled, the layer data is treated as a floating selection. Transparency data will be presented

via the mask portion of the interface rather than with the input data.

### inputHandling & outputHandling

The *inputHandling* and *outputHandling* fields specify the pre–processing and post–processing actions on the image data respectively.

**Table 11-14: FilterCaseInfo handling modes**

| Handling mode | Description |
|---|---|
| 0=inCantFilter = outCantFilter | indicates that this case is not supported by the plug–in filter |
| 1=inStraightData = outStraightData | indicates that the plug–in filter does not expect the plug–in host to do anything to the image data. |
| *The next three modes are matting cases, which are useful when performing distortions and blurs.*<br>You can matte the data, process it, and then dematte to remove the added color.<br>For these cases, the matting is defined as follows:<br>`mattedValue = ((unmattedValue * transparency) + 128) / 255 +`<br>`((matConstant * (255 - transparency)) + 128) / 255`<br>Dematting is defined as follows:<br>`unmattedValue = ((mattedValue - matConstant) ./ transparency) + matConstant`<br>with the `./` operator defined as an 8 bit fixed–point divide and the result value=0...255. | |
| 2=inBlackMat = outBlackMat | For input, matte the image data with black=0 values based on the transparency.<br>For output, dematte the image data using black (=0) values. |
| 3=inGrayMat = outGrayMat | Matte the image data with gray (=128) values based on the transparency on input. Dematte the image data using gray values on output. |
| 4=inWhiteMat = outWhiteMat | Matte the image data with white (=255) values based on the transparency on input. Dematte the image data using white values on output. |
| *Input-only related modes* | |
| 5=inDefringe | Defringe transparent areas filling with the nearest defined pixels using taxicab distance. Note that this only applies to fully transparent pixels. |
| 6=inBlackZap | Set color component of totally transparent pixels to black. |
| 7=inGrayZap | Set color component of totally transparent pixels to gray. |
| 8=inWhiteZap | Set color component of totally transparent pixels to white. |
| 10=inBackgroundZap | Set color component of totally transparent pixels to the current background color. |
| 11=inForegroundZap | Set color component of totally transparent pixels to the current foreground color. |
| *Output-only related modes* | |
| 9=outFillMask | This mode results in the transparency mask automatically being filled with full opacity in the area affected by the filter. This is only valid for the editable transparency cases. This option is provided to make it easy to write a plug–in similar to Photoshop's Clouds plug–in, which fills an area with a value. |

**Table 11-15: FilterCaseInfo flags1 parameters**

| Field | Values |
|---|---|
| 0=PIFilterDontCopyToDestinationBit | 0=copySourceToDestination<br>1=doNotCopySourceToDestination |
| Normally source data is copied to the destination before filtering. This degrades performance for filters which write all the output pixels. Setting this bit inhibits copying. | |
| 1=PIFilterWorksWithBlankDataBit | 0=doesNotWorkWithBlankData<br>1=worksWithBlankData |
| Determines whether the filter will work on “blank” areas that are completely transparent. If not, an error message will be given when the filter is invoked on a blank area. This is only valid for the editable transparency case because that is the only case where you could create opacity—in the protected transparency case, you would be left with what you started with: completely blank data. | |
| 2=PIFilterFiltersLayerMaskBit | 0=doesNotFilterLayerMasks<br>1=filtersLayerMasks |
| In cases where transparency is editable, this flag determines if Layer Masks are filtered. (See the “Add Layer Mask” item in the Layers palette menu to create a layer mask.) Setting this bit adds the layer mask to the set of target channels if: transparency for the layer is editable (i.e., this is one of the editable transparency cases), the bit is set, and the layer mask is specified as being positioned relative to the layer rather than the image in Layer Mask Options. The distinction based on position is based on the assumption that layer relative masks are distorted with the layer; image relative masks are independent of the layer. | |
| 3=PIFilterWritesOutsideSelectionBit | 0=doNotWriteOutsideSelection<br>1=writeOutsideSelection |
| In the image with selection and layer with selection cases, does the filter want to write beyond the confines of the selection? (This is generally rude but in some cases it’s better than the alternatives. If you use this, be sure to support layer transparency data as an alternate mask.) | |

**Note:** This field is not a `FlagSet`. The first bit, `PIFilterDontCopyToDestinationBit`, is in the least–significant bit of the flag byte.

### flags2

The *flags2* field of the `FilterCaseInfo` structure is reserved, and should be zero.

## Format–specific properties

These properties are only applicable to format plug–in modules.

**Table 11-16: Adobe Photoshop format-specific properties**

| Type | Name | Key | Description |
|---|---|---|---|
| TypeCreator-Pair | PIFmtFileTypeProperty | 0x666d5443L ('fmTC') | Default type and creator code used for files newly created with this format plug–in.<br>Under Windows, files don't store TypeCreator information, except internally, so this property is not required; they are always interpreted as of type 'BINA' and creator 'mdos'.<br>All the info regarding what files can be read and written is obtained from the PIReadExtProperty or the PIFilteredExtProperty.<br>Under Windows, PiMI extensions are converted to PIReadExtPropertys, so use of PIFilteredExtProperty requires additional coding if you are porting a 16–bit plug–in format module to 32–bit. |
| Array of TypeCreator-Pair | PIReadTypesProperty | 0x52645479L ('RdTy') | List of type and creator pairs which the format plug–in can read. Specifying a value of four spaces (0x20202020L) matches any type or creator. |
| Array of TypeCreator-Pair | PIFilteredTypesProperty | 0x66667454L ('fftT') | List of type and creator pairs for which the file format plug–in should be called to determine if the file can be read. Specifying a value of four spaces (0x20202020L) matches any type or creator. |
| Array of OSTypes | PIReadExtProperty | 0x52644578L ('RdEx') | List of extensions which the format plug–in can read. The extension is stored in the first three characters of the OSType. The fourth character must be a space. |
| Array of OSTypes | PIFilteredExtProperty | 0x66667445L ('fftE') | List of extensions for which the file format plug–in should be called to determine if the file can be read. |

**Table 11-16: Adobe Photoshop format-specific properties (Continued)**

| Type | Name | Key | Description |
|---|---|---|---|
| FlagSet | PIFmtFlagsProperty | 0x666d7466L ('fmtf') | This property contains a set of flags which control the operation of file format plug–ins. The default value for any flag is FALSE. See table 11-17. |
| Point | PIFmtMaxSizeProperty | 0x6d78737aL ('mxsz') | The maximum number of rows and columns that can be in an image saved in this format. Photoshop will use this field to screen out ineligible formats. |
| Array of int16s | PIFmtMaxChannelsProperty | 0x6d786368L ('mxch') | An array of counts of the maximum number of channels which can/will be saved for a given image mode.<br>This array is indexed by the plug–in mode constants. For example, if your format plug–in supports a single alpha channel in RGB mode, you should set maxChannels [plugInModeRGBColor]=4.<br>A plug–in may still be asked to save more channels than it reports it can support. This field exists primarily so that Photoshop can warn the user that alpha channels will be discarded. |

**Table 11-17: PIFmtFlagsProperty parameters**

| Field | Description |
|---|---|
| 0=PIFmtReadsAllTypesFlag | Obsolete. |
| 1=PIFmtSavesImageResourcesFlag | Resources besides image data, such as printing information, pen tool paths, etc.. are known as image resources. The plug–in format has the option of taking responsibility for these resources by reading and writing a block of data containing the image resources. If FALSE, Photoshop will add the image resources to the file's Mac OS resource fork but this will not be portable to other platforms. |
| 2=PIFmtCanReadFlag | =TRUE if the file format can read files. |
| 3=PIFmtCanWriteFlag | =TRUE if the file format can write files. |
| 4=PIFmtCanWriteIfReadFlag | Whether plug–in can write the file if the plug–in originally read the file. |

## Scripting–specific properties

**Table 11-18: Adobe Photoshop scripting-specific properties**

| Type | Name | Key |
|---|---|---|
| 80-bytes + string | HasTerminology | 0x6873746DL ('hstm') |

This property indicates whether an 'aete' resource is present and whether your plug-in is scripting-aware for Photoshop and AppleScript.

```
typedef struct HasTerminology
{
      int32    classID;      // classID from 'aete'
      int32    eventID;      // eventID from 'aete' or NULL if none
      int16    aeteResNum;   // number of 'aete' resource
      CString  uniqueID;     // unique ID string (UUID or your own ™/©). If present,
                                ignores AppleScript and keeps local to Photoshop.
} HasTerminology;
```

All scripting-aware plug-ins must have an 'aete' dictionary resource. aeteResNum should be the resource number for that dictionary. Multiple dictionaries are usually provided when a file contains more than one plug-in.

Scripting-aware Filters, Selection, and Color Picker modules must provide both a classID and an eventID. All other types of plug-ins must provide a classID and pass typeNull='null' for the eventID.

**Note:** If a non-filter does not pass typeNull for eventID, then errors will occur, as the existence of an eventID triggers the host to parse the dictionary as if it was for a Filter, Selection, or Color Picker module.

**Ignoring AppleScript**
If you don't care about AppleScript compatibility, supplying the uniqueID string automatically makes your plug-in's scripting scope to only the host. Any external AppleScript or similar calls to your plug-in will be ignored.

## Adobe Photoshop PiPL Syntax

This information is included as reference material. If you use the example source code and the documentation included on the Photoshop SDK, you probably won't need to worry about the specifics of the PiPL syntax.

```
# Miscellaneous definitions

<OSType>

<int16>

<int32>

<epsilon> :=

# Beginning of real grammar.

<PiPL spec> := <resource header> <resource body>

<resource header> :=
    "resource" "'PiPL'" "("
     <resourceID> <optional resource name> <optional attribute list>
    ")"

<optional name> :=
    <epsilon> |
    "," <string>

<optional attribute list> :=
    <epsilon> |
    "," <attribute> <attribute list tail>

<attribute list tail> :=
    <epsilon> |
     "|" <attribute> <attribute list tail>

<resource body> :=
    "{" "{"
    <property list>
    "}" "}"

<property list tail> :=
    <epsilon> |
    "," <property> <property list tail>

<property list> :=
    <epsilon>
    | <property> <property list tail>

<property> :=
    <kind property> |
    <version property> |
    <priority property> |
    <required host property> |
    <name property> |
    <category property> |
    <68k code descriptor property> |
    <powerpc code descriptor property> |
    <win32 x86 code property> |
    <supported modes property> |
```

```
    <filter case info property> |
    <format file type property> |
    <read types property> |
    <write types property> |
    <filtered types property> |
    <read extensions property> |
    <write extensions property> |
    <filtered extensions property> |
    <format flags property> |
    <format maximum size property> |
    <format maximum channels property> |
    <parsable types property> |
    <parsable extensions property> |
    <filtered parsable types property> |
    <filtered parsable extensions property> |
    <parsable clipboard types property>

<kind property> := "Kind" "{" <kind ID> "}"

<kind ID> := <OSType> |
    "Filter" |
    "Parser" |
    "ImageFormat" |
    "Extension" |
    "Acquire" |
    "Export"

<version property> := "Version" "{" <version clause> "}"

<version clause> := <int32> |
    "(" <wired version ID high> "<<" "16" ")" "|"
    "(" <wired version ID low> ")" |
    <wired version ID>

<wired version ID> := "FilterVersion" |
    "ParserVersion" |
    "ImageFormatVersion" |
    "ExtensionVersion" |
    "AcquireVersion" |
    "ExportVersion"

<wired version ID high> := "latestFilterVersion" |
    "latestParserVersion" |
    "latestImageFormatVersion" |
    "latestExtensionVersion" |
    "latestAcquireVersion" |
    "latestExportVersion"

<wired version ID high> := "latestFilterSubVersion" |
    "latestParserSubVersion" |
    "latestImageFormatSubVersion" |
    "latestExtensionSubVersion" |
    "latestAcquireSubVersion" |
    "latestExportSubVersion"

<priority property> := "Priority" "{" <int16> "}"

<required host property> := "Host" "{" <OSType> "}"

<name property> := "Name" "{" <string> "}"

<category property> := "Category" "{" <string> "}"
```

```
<68k code descriptor property> := "Code68k" "{" <OSType>, <int16> "}"

<powerpc code descriptor property> := "CodePowerPC" "{"
    <int32>, <int32> <optional name> "}"

<win32 x86 code property> := "CodeWin32X86" "{" <string> "}

<bitmap support> := "noBitmap" | "doesSupportBitmap"
<grayscale support> := "noGrayScale" | "doesSupportGrayScale"
<indexed support> := "noIndexedColor" | "doesSupportIndexedColor"
<RGB support> := "noRGBColor" | "doesSupportRGBColor"
<CMYK support> := "noCMYKColor" | "doesSupportCMYKColor"
<HSL support> := "noHSLColor" | "doesSupportHSLColor"
<HSB support> := "noHSBColor" | "doesSupportHSBColor"
<multichannel support> := "noMultichannel" | "doesSupportMultichannel"
<duotone support> := "noDuotone" | "doesSupportDuotone"
<LAB support> := "noLABColor" | "doesSupportLABColor"

<supported modes property> := "SupportedModes"
    "{"
    <bitmap support> ","
    <grayscale support> ","
    <indexed support> ","
    <RGB support> ","
    <CMYK support> ","
    <HSL support> ","
    <HSB support> ","
    <multichannel support> ","
    <duotone support> ","
    <LAB support>
    "}"

<filter case info property> := "FilterCaseInfo"
    "{"
        "{"
        <filter info case> # filterCaseFlatImageNoSelection
        <filter info case> # filterCaseFlatImageWithSelection
        <filter info case> # filterCaseFloatingSelection
        <filter info case> # filterCaseEditableTransparencyNoSelection
        <filter info case> # filterCaseEditableTransparencyWithSelection
        <filter info case> # filterCaseProtectedTransparencyNoSelection
        <filter info case> # filterCaseProtectedTransparencyWithSelection
        "}"
    "}"

<filter info case> :=
    <input matting> "," <output matting> ","
    <layer mask flag> "," <blank data flag> "," <copy source flag>

<input matting> :=
    "inCantFilter" |
    "inStraightData" |
    "inBlackMat" |
    "inGrayMat" |
    "inWhiteMat" |
    "inDefringe" |
    "inBlackZap" |
    "inGrayZap" |
    "inWhiteZap" |
    "inBackgroundZap" |
    "inForegroundZap"

<ouput matting> :=
```

```
    "outCantFilter" |
    "outStraightData" |
    "outBlackMat" |
    "outGrayMat" |
    "outWhiteMat" |
    "outFillMask"

<layer mask flag> := "doesNotFilterLayerMasks" | "filtersLayerMasks"
<blank data flag> := "doesNotWorkWithBlankData" | "worksWithBlankData"
<copy source flag> := "copySourceToDestination" |
    "doNotCopySourceToDestination"

<type creator pair> :=
    <OSType> "," <OSType>

<format file type property> :=
    "{"
    <type creator pair>
    "}"

<type creator pair list tail> :=
    <epsilon> |
    "," "{" <type creator pair> "}" <type creator pair list tail>

<type creator pair list> :=
    <epsilon> |
    "{" <type creator pair> "}" <type creator pair list tail>

<read types property> :=
    "{"
    <type creator pair list>
    "}"

<write types property> :=
    "{"
    <type creator pair list>
    "}"

<filtered types property> :=
    "{"
    <type creator pair list>
    "}"

<ostype list tail> :=
    <epsilon> |
    "," "{" <OSType> "}" <ostype list tail>

<ostype list> :=
    <epsilon> |
    "{" <OSType> "}" <ostype list tail>

<read extensions property> :=
    "{"
    <ostype list>
    "}"

<write extensions property> :=
    "{"
    <ostype list>
    "}"

<filtered extensions property> :=
    "{"
```

```
    <ostype list>
    "}"

<saves image resources flag> :=
    "fmtDoesNotSaveImageResources" | "fmtSavesImageResources"

<can read flag> :=
    "fmtCannotRead" | "fmtCanRead"

<can write flag> :=
    "fmtCannotWrite" | "fmtCanWrite"

<write if read flag> :=
    "fmtWritesAll" | "fmtCanWriteIfRead"

<format flags property> :=
    "{"
    <saves image resources flag> ","
    <can read flag> ","
    <can write flag> ","
    <write if read flag>
    "}"

<format maximum size property> :=
    "{"
    <int16>, <int16>
    "}"

<format maximum channels property> :=

<parsable types property> :=
    "{"
    <type creator pair list>
    "}"

<parsable extensions property> :=
    "{"
    <ostype list>
    "}"

<filtered parsable types property> :=
    "{"
    <type creator pair list>
    "}"

<filtered parsable extensions property> :=
    "{"
    <ostype list>
    "}"

<parsable clipboard types property> :=
    "{"
    <ostype list>
      "}"
```

# 12. Adobe Premiere

This chapter describes properties and useful resources of Adobe Premiere for creating plug-ins that work under multiple applications.

**Table 12-1: Adobe Premiere version and signature information**

| Description | Value |
|---|---|
| Mac OS version | 4.2 |
| Mac OS release date | 8/1/95 |
| Windows version | 4.0 |
| Windows release date | 5/1/95 |
| Backward-compatibility targets Mac, Win | 4.0, 4.0 |
| Signature | `'PrMr'` |

# Adobe Premiere and Adobe Photoshop

**Table 12-2: Adobe Premiere emulating Photoshop host**

| Description | Value |
|---|---|
| Signature | '8BIM' |
| Host version support Mac, Win | 2.5, 3.0 |
| Required adaptor | N/A |
| Resource | 'PiMI' |
| Supported module types | Filter |
| Plug-in folder default | Adobe Premiere 4.2/Adobe Premiere Plug-ins |
| Plug-in aliases | Premiere does not resolve folders. |
| Plug-in load order | Loads references and code on launch. |
| *How to access the different plug-ins while using Adobe Premiere:* | |
| Filter modules | **Clip»Filters** |

### Host version support

Adobe Premiere emulates the Adobe Photoshop 2.5 Plug-in API. All of the 2.5 API calls and functions are implemented, except:

1. Any callbacks related to Acquire, Export or Format modules
2. Any 3.0+ callback services or suites
3. If your plug-in is 24-bit pixels (RGB) vs. 32-bit (RGB+alpha), renumber your PiMI major version number to ≤ 3, rather than ≥ 4.

**Note:** Because hosts like Adobe Premiere implement a version of the Adobe Photoshop plug-in API that is earlier then the current version, it is very important you check for validity and existence of suite versions and their callbacks *before* you use them.

### Creating dynamic resources

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

## Numerics



## A



## B



## C

## D



## E



## F

## G



## H



## I



## K



## M



## N



## O



## P

## R



## S



## T



## V



## W