Adobe

Adobe After Effects™ version 3.1

Software Development Kit

release 2 for Macintosh

Revised: 13 November 1996

Adobe After Effects 3.1 Software Development Kit r2

Most of the material for this document was derived from work by Russell Belfer, David Herbstman, David Simons, and Daniel Wilk. It was then compiled, edited, and reformatted into its current form by Brian Andrews.

Version History		
January 1993	Russell Belfer	Version 1.0 – Initial SDK release.
January 1994	Dan Wilk	Version 2.0 - Updates.
August 1994	Dave Herbstman Dan Wilk	Version 2.0.1 – PowerPC Update.
5 March 1996	Brian Andrews	Version 3.0 – Preliminary release for the After Effects developer kitchen.
21 June 1996	Brian Andrews	Version 3.1 – Final 3.x release.
13 November 1996	Brian Andrews	Version 3.1r2 - Minor updates.

1 Introduction 5

How to Use This Guide 5

About This Guide 6

Changes Since the Last Release 7

2 Plug-In Overview 8

The After Effects Pipeline 8

Key to the Diagram 9

Where Plug-ins Are Found 10

How Plug-ins Are Invoked. 10

Writing Plug-ins 12

Command Selectors 12

Global Commands 13

Sequence Commands 14

Frame Commands 14

User Interface Command 15

3 Plug-In Resources 16

4 Effects Filters. 17

Effect Input. 17

PF_InData Structure 17

PF_ParamsList Array of Parameter Descriptions 20

Effect Output 22

PF_OutData Structure 22

PF_OutFlags 24

PF_LayerDef Structure 27

Callbacks. 28

User Interaction Related Callbacks 28

Kernel Flags 30

Graphics Utility Callbacks 31

Intrinsic Callbacks 37

ANSI Callbacks 38

Colorspace Conversion Callbacks 39

5 Custom User Interface 41

Getting UI Events. 41

PF_Context Structure 42

Event Types (PF_EventType) 43

Event Unions (PF_EventUnion) 44

Click Event 44

Draw Event 44

Key Down Event 45

Effect Window Information (PF_EffectWindowInfo) 45

UI Callbacks (PF_EventCallbacks) 46

6 Input and Output Plug-Ins 48

Resource Structures 48

‘PiPL’ Resource Structure 48

‘FXMF’ PiPL Atom 48

Interface Record Structure 49

Time Extension Structures 49

Calling Sequences 52

Other notes 53

7 Special Considerations 54

Extent Rects 54

Alpha Channels 55

Parameter Situations 55

Sequence Data 55

Error Handling 56

Debugging 56

Be Responsive 56

Adding Parameters 56

Index 58

1 Introduction

Welcome to the Adobe After Effects 3.1 Software Development Kit, release 2, for the Macintosh! Adobe After Effects supports plug-in modules for applying special effects to images and for performing file input and output. Effects modules can appear in the Effect menu or the render queue of the program, and the controllers (sliders, pop-ups, etc.) that configure the effect appear in an Effect floater. File I/O modules appear in the File menu. This document describes the programming interface to plug-in effects along with new custom user interface elements. It also describes how Adobe After Effects supports the Adobe Photoshop plug-in interface for file input/output and effects filters.

The interface to Adobe After Effects plug-ins is somewhat similar to the one used in Adobe Photoshop, authors of Photoshop filters should be able to get up to speed quickly with After Effects plug-ins. However, just as Photoshop plug-ins were completely different in detail from the Silicon Beach plug-ins that preceded them (see the historical note in “Writing Plug-in Modules for Adobe Photoshop™,” by Thomas Knoll), so too, After Effects’ plug-ins are completely different from Photoshop’s. We add support for time-varying parameters, application integrated plug-in parameter control, and a library of graphical utility callbacks, and we eliminate the virtual memory complications and varying image data representations.

This guide assumes that you are proficient in C language programming and tools. The source code files in this toolkit are written for the Metrowerks CodeWarrior software development environment.

You should have a working knowledge of Adobe After Effects and understand how plug–in modules work from a user’s viewpoint. This guide assumes you understand After Effects and basic video editing terminology.

How to Use This Guide

There are four places to look to learn about Adobe After Effects plug-ins.

- First, you should already be familiar with the Adobe After Effects program — if not, you should review the After Effects manual and/or the new *Adobe After Effects Classroom in a Book*. Understanding movies, time-varying parameters, alpha channel, and the After Effects model of applying effects is vital to understanding this plug-in spec.

- Second, this SDK Guide gives an overview and much specific information about writing plug-ins. A quick read of this document is a good next step once you have some knowledge of the program itself.

- Third, the header files are extensively commented. To fully understand the plug-in spec, you will have to become at least mildly familiar with those files. Hopefully you will find them quite readable. Much of the information in the headers is also in this document, but some details are presented only in the header files.

The header files that describe the Adobe After Effects plug-in specification are AE_Effect.h, AE_EffectCB.h, and AE_EffectUI.h. The letters 'PF' (at one point in After Effects' development cycle effects were called "filters", hence the PF) and an underscore are used as a prefix for constants, types, macros, and routines defined by in the header files. For example, rather than defining a type ParameterBlock, we would call it PF_ParameterBlock. This will make it easy to distinguish the definitions that come from our header files from those that you make.

- Fourth, the sample code is the best place to start when actually writing a plug-in (at least for beginners). It's easier to modify than to create from scratch, and the samples provide working examples of some of the detailed aspects of the spec that can be hard to understand the first few times through.

This toolkit documentation starts with information that is common to all the plug-in types. The rest of the document is broken up into chapters specific to plug-in types.

Chapter 2 presents a Plug-In Overview which describes the attributes common to all After Effects plug-ins such as where they are found, how they are invoked, and the command selectors. This information is essential to writing After Effects plug-ins and is the section you should read next.

Chapter 3 points to information about the Plug-In Resources PiPL (Plug-In Property List) and ANIM. Starting with After Effects 3.0, all plug-ins communicate their attributes to After Effects using the PiPL resource. ANIM, which is a type of PiPL atom supported exclusively by After Effects, allows Photoshop Plug-ins to have a time based component so their effect can be varied over time. This information is also essential and should be read following chapter 2.

Chapter 4 dives into the details of writing Effects Filters. This is the bulk of the contents of this SDK and will be applicable to most After Effects plug-in developers.

Chapter 5 describes how to incorporate Custom User Interface elements into your plug-ins. This information is optional and is of interest if you wish to extend and customize the look of your plug-ins.

Chapter 6 describes how to write Input and/or Output Plug-Ins. This information is optional and only applicable to developers wishing to write plug-ins to add file formats not supported by the core After Effects program.

Finally chapter 7, Special Considerations, contains some pointers and advice on how to deal with parts of the After Effects plug-in API which are especially tricky. This is recommended reading after looking at chapter 2, 3, and 4.

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 font family is used throughout the manual with Courier used for code examples.

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

Changes Since the Last Release

This is the second release of the After Effects 3.1 SDK for Macintosh. It contains a few bug fixes and documentation clarifications. It also identifies a common parameter problem reported by several developers, and offers the solution. See the new section "Adding Parameter" in chapter 7 for the details.

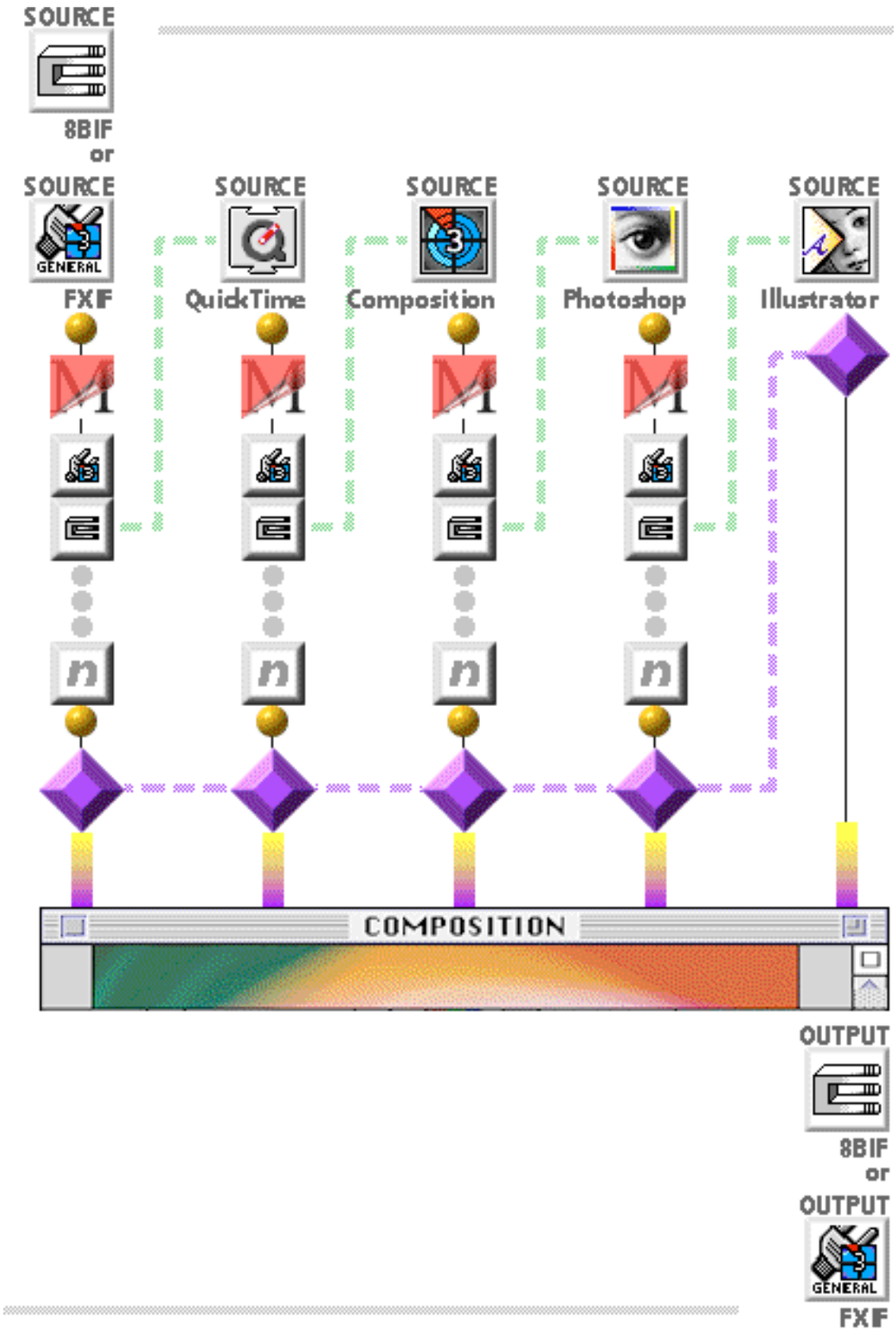
2 Plug-In Overview

This chapter should get you started in understanding how After Effects plug-ins operate. This information is universal to all the After Effects plug-in types.

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

The After Effects Pipeline

The following diagram depicts the After Effects pipeline and where plug-ins fit into the picture. There is a key following the diagram which explains some of the symbols.



Key to the Diagram

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

Mask – Bezier shapes with feathering are matted with an existing alpha channel.

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

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

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

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

Layer Parameters

Track Matte

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

FXIF – This is an After Effects I/O plug-in. This is essentially an 8BIF plug-in which has been extended for After Effects. These are described in chapter 6 of this document.

Where Plug-ins Are Found

When launched, After Effects looks for plug-ins in all sub-folders of the folder where the application itself resides. It will recursively descend into all folders up to 10 levels deep — aliases will also be traversed. This lets the user organize the plug-ins however they want and keep aliases to folders of plug-ins. Folders with names surrounded in parentheses (e.g. ‘(old plug-ins)’) or proceeded by the symbol ¬ are not scanned.

To identify After Effects and Photoshop plug-ins, After Effects now looks for a PiPL (or Plug-in Property List) resource. This resource (which is the subject of the next chapter) describes the various attributes and capabilities the plug-in supplies. The author is free to include any other resources in the plug-in file for use by the plug-in.

How Plug-ins Are Invoked

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

After Effects can view a plug-in with different scopes. If the user applies the same effect in multiple places throughout the project, After Effects creates multiple “instances” of the effect in the model. Each instance is responsible

for a sequence of single frames over time. All instances of the effect can share global data. Each instance can share data between all frames in its sequence. Ultimately, the effect is invoked to render a single frame of data. (There is an interesting parallel to the C++ language here: each effect defines a class, each application of the effect creates an instance, and a specific method of that instance is invoked to render each single frame.)

Now is a good time to provide a short explanation of a term you will see later on: flattening handles. When Adobe After Effects writes a project with sequence data out to disk, it records each plug-in's sequence specific data into the file. In each case the data is assumed to be a handle that you have allocated (if it is not NULL) containing your information. If you have other pointers or handles within that data block, when the data is read back in from disk, those pointers will be invalid. The data block is not simply one contiguous block of information. Because this type of complex structure is sometimes unavoidable, After Effects has a notion of flattening sequence shared data before recording it to disk. When the effect is asked to flatten a handle, it should make all information into one contiguous block from which it can later recover the old structure. When this block is read in from disk, the effect will be given the opportunity to unflatten the data before it is expected to do any rendering work.

To invoke an effect, Adobe After Effects looks at the PiPL to find the entry point of the code. Every plug-in has a single entry point with selectors indicating the desired operation.

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

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

The cmd parameter is a selector for the requested plug-in function.

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

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

The params parameter points to an array of structures that describe the values of the plug-in's parameters at the time of the current invocation. This first of the parameters (params[0]) describes the input image upon which the effect is to act. These params settings will only be valid for certain function selectors and are set by the application user to configure the effects of the plug-in.

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

The extra parameter is new in 3.x and is used by plug-ins incorporating a custom user interface. This will be described in chapter 4 and 5 for the callbacks which use it.

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

Writing Plug-ins

The easiest way to write plug-ins is to modify already existing plug-in source code. There are many details of the After Effects plug-in spec with which the beginning effect writer need not be concerned. We provide a couple of sample effects from which you can build your own. In particular, you will probably want to duplicate our main selector dispatch routine and copy and paste together pieces of the parameter definition sub-routine.

On 68K based Macintoshes, plug-ins are code resources. This causes a variety of problems with global variables, with multi-segment code resources, and with jump tables. We recommend using Metrowerks CodeWarrior for plug-in development, setting up the A4 register for global accesses (if you need them at all) and using CodeWarrior's EnterCodeResource()/ ExitCodeResource() protocol. If you feel you have a deep understanding of the nature of code resources, you are welcome to write plug-ins using MPW, or any other development system you like (provided you can access functions with C calling conventions). We provide plug-ins with a pointer to the QuickDraw globals in case the plug-in needs access to them.

On PowerPC based Macintoshes, plug-ins are code fragments and thus avoid the issues described above.

Adobe After Effects plug-ins can assume a 68020 or faster processor and System 7.0 or later (and thus 32-bit QuickDraw). Unlike the previous version of After Effects, version 3.x will run with or without QuickTime installed. The application also tells the plug-in whether there is a math co-processor, and there are certain conventions for plug-ins that require one (sample code will be given). After Effects 3.x requires an FPU, this isn't likely to change, but it's a good idea to check anyway.

Command Selectors

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

The Adobe After Effects command selectors are:

```
PF_Cmd_ABOUT
PF_Cmd_GLOBAL_SETUP
PF_Cmd_GLOBAL_SETDOWN
PF_Cmd_PARAMS_SETUP

PF_Cmd_SEQUENCE_SETUP
PF_Cmd_SEQUENCE_RESETUP
PF_Cmd_SEQUENCE_FLATTEN
PF_Cmd_SEQUENCE_SETDOWN

PF_Cmd_DO_DIALOG
PF_Cmd_FRAME_SETUP
PF_Cmd_RENDER
PF_Cmd_FRAME_SETDOWN
PF_Cmd_PARAMS_UPDATE

PF_Cmd_EVENT
```

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

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

Next we examine the selectors in a little more detail.

Global Commands

PF_Cmd_ABOUT

When the effect gets this command, it should display an information dialog box about the effect module. The easiest thing to do is use the PF_SPRINTF callback (see Callbacks in the next chapter) to write the info into the out_data->return_msg field (see PF_OutData description in the next chapter). Adobe After Effects will bring up a simple undecorated modal dialog with your About... text proudly displayed. Please include the version number of your effect in the abort dialog. The About command could be sent at any time, so the effect cannot use global data or anything else. (Except, as always, the current resource file will be set to your effects module.)

PF_Cmd_GLOBAL_SETUP

When you get this command, you should set any of the necessary output flags or PF_OutData fields (described in the next chapter). You should also set the my_version field to the version of your plug-in.

PF_Cmd_GLOBAL_SETDOWN

You should free any global data you have allocated when you get this command.

PF_Cmd_PARAMS_SETUP

Here you should describe any parameters your effect uses by invoking the PF_ADD_PARAM callback described below. This selector is sent after global setup. It also describes to After Effects any kind of custom UI elements you may be using.

Sequence Commands

PF_Cmd_SEQUENCE_SETUP

This is sent when the effect is first applied to a layer. A sequence is a series of images that will all be of the same size and in the same context. You can allocate sequence data at this time — many of the PF_InData input fields are defined at this time. See the PF_InData description in the next chapter.

PF_Cmd_SEQUENCE_RESETUP

This is sent when something significant about the PF_InData changes — for instance, the input image size is altered. The parameters that the user has set are still being applied; this message gives the effect a chance to re-adjust sequence data to the new sequence information if that is needed. Also, this selector is passed as the opportunity to unflatten flat sequence data when a project is first read in from disk — see the example code for unflattening sequence data.

PF_Cmd_SEQUENCE_FLATTEN

This selector is passed to flatten unflat sequence data so it can be written to disk. See the PF_OutFlag_SEQUENCE_DATA_NEEDS_FLATTENING flag below. After the data is flattened, free the unflat data and set the out_data->sequence_data to the new flat data.

PF_Cmd_SEQUENCE_SETDOWN

You should free any sequence data you have allocated when you get this command. After the memory has been freed, you should set the in_data and out_data pointers to null.

Frame Commands

PF_Cmd_DO_DIALOG This command indicates the Options button or a menu command has been selected and the effect should bring up its options dialog. This command will only be sent it the effect has indicated that it has an options dialog with PF_OutFlag_I_DO_DIALOG flag (described in the next chapter).

PF_Cmd_FRAME_SETUP

This is sent immediately before each frame is invoked. You can allocate frame data at this time, if you wish, or you can just wait for the RENDER which will immediately follow. If your effect changes the size of its output buffer (like the Drop Shadow effect), you must specify your output height, width, and relative origin at this command. All of your parameters except the input layer will be valid at this time. If

you set width and height to 0, the After Effects rendering pipeline will ignore this scope altogether.

PF_Cmd_RENDER

This is the call to render the frame. All fields in the PF_InData will be valid at this time and you can inquire parameters or what-have-you. This should set the output frame with the new image data. This is the main action command. This will occur once in After Effects for each time the FRAME_SETUP selector is called, except when you resize the output buffer to be (0,0). If this selector does not complete before the user interrupts, the rendering will be canceled and the rendering results will not be used.

PF_Cmd_FRAME_SETDOWN

If you allocated frame data in PF_Cmd_FRAME_SETUP, this is the time to free it and clean up after rendering the frame.

PF_Cmd_PARAMS_UPDATE

In After Effects 3.0 this command is never sent. This command would be sent if you set the PF_OutFlag_SEND_PARAMS_UPDATE flag (see below). This would allow you to modify the params array contents to change the display of parameters. This command would be sent whenever the user interacts with the parameter controllers. Changing param values in this way is called "parameter supervision."

User Interface Command

PF_Cmd_EVENT

This command makes use of the extra parameter mentioned above. This is a command to handle an event indicated by the event type field, which is a member of the structure pointed to by extra. See chapter 5 for further information.

3 Plug-In Resources

As mentioned earlier, as of release 3.0 After Effects plug-ins must include a PiPL (Plug-in Property List) resource. This was taken from Photoshop which originated the PiPL resource. There are two major aspects of PiPL in which to be aware.

- All After Effects plug-ins must include a PiPL resource which identifies the plug-in and provides flags and other static properties that control the operation of the plug-in.

- While After Effects can load and apply Photoshop 3.0.4 or earlier effects filters, the filter settings are static. That is, when applying the filter to an object, the filter settings cannot be altered over time. However, Photoshop effects filters can be made After Effects savvy, which allows their settings to be altered, or dynamically interpolated over time, by adding an ANIM property to their PiPL resource.

For detailed information on both of these topics, please refer to the companion document entitled *Adobe Graphic Application Products, Cross-Application Plug-in Development Resource Guide*. Chapters 3 and 4 of this document cover After Effects specific information.

As you look at the sample plug-ins included with this SDK you be able to see how its PiPL is constructed. It's highly recommended you use a resource editing program like *Resorcerer* along with the supplied PiPL template for viewing the examples. ResEdit cannot visually edit a PiPL resource.

4 Effects Filters

This chapter gets into the details of writing an effects filter. While reading this material you might want to have the SDK sample code and headers alongside as there are several effects plug-ins which demonstrate many of these concepts.

There are two structures each for input and output. The PF_InData parameter block and the PF_ParamsList array of parameter descriptions are the input to the effect. The PF_OutData parameter block and the output image PF_LayerDef are the output from the effect. These are described next.

Effect Input

The input parameters of an effect are the image to which it is applied, and any sliders, pop-ups, check boxes, angles, points, colors, or whatever other parameters the effect defines that the user can control and manipulate over time. These parameters are passed in the PF_ParamsList array of structures. Everything else is passed in the PF_InData structure.

PF_InData Structure

Here is the format of the PF_InData structure, a description of the fields follow. Please note that this structure is read-only, After Effects will ignore any changes you may make to it.

```
typedef struct {
   PF_InteractCallbacks    inter;
   void                    *utils;
   PF_ProgPtr              effect_ref;
   PF_Quality              quality;
   PF_SpecVersion          version;
   long                    serial_num;
   long                    appl_id;
   long                    num_params;
   long                    reserved;
   long                    what_cpu;
   long                    what_fpu;
   PF_QDGlobals            *qd_globals;
   long                    current_time;
   long                    time_step;
   long                    total_time;
   long                    local_time_step;
   long                    time_scale;
   PF_Field                field;
   PF_Fixed                shutter_angle;
   long                    width;
   long                    height;
   Rect                    extent_hint;
   long                    output_origin_x;
   long                    output_origin_y;
   PF_RationalScale        downsample_x;
   PF_RationalScale        downsample_y;
   PF_RationalScale        pixel_aspect_ratio;
   PF_InFlags              in_flags;
   Handle                  global_data;
```

```
   Handle                  sequence_data;
   Handle                  frame_data;
} PF_InData;
```

On receiving any given PF_Cmd, only certain fields in the input block will have valid values. Each field described below tells when it is valid. The fields of this structure represent the following:

inter — This is a structure containing callbacks related to user interaction. This structure contains the callbacks to add parameters, to check if the user has interrupted the effect, to display a progress bar, and to inquire parameter values at times other than the current time being rendered. The details of these callbacks are given below in the Callbacks section. When each callback can validly be executed is described there.

utils — This is a pointer to a block of useful graphical and mathematical callbacks provided for the effects module. The definition of this block is in the AE_Effect.h file. This field is a void *, which can be confusing. See AE_EffectCB.h for macros to use these functions. This pointer will be defined at all times. The descriptions of these utility callbacks are below in the Callbacks section.

effect_ref — This is an opaque piece of data that needs to be passed to most of the various callback routines. Don't worry about it. (Be happy about it.)

quality — This is set to one of the PF_Quality constants (PF_Quality_HI or PF_Quality_LO) to describe the Quality currently chosen by the user. Ideally, your effect should do a faster version with LO quality, and a better, "broadcast"-able version with HI quality. Also, some of the graphics utility callbacks perform differently between LO and HI quality. This field is defined for all PF_Cmds related to SEQUENCE and FRAME (obviously, including RENDER).

version — This is the version of the effects spec with which you are being invoked. This will be set in GLOBAL_SETUP, in which you specify the spec version you need to run successfully and the version you would like to being run with.

serial_num — This is the serial number of the invoking application. Use this information as you desire.

appl_id — This is the identifier of the invoking application. It will be the creator long of the app — for Adobe After Effects that is 'FXTC'.

num_params — This is set to the number of input parameters you are receiving.

what_cpu — This is set to the return value from Gestalt (see Inside Mac, Vol VI) asking what sort of CPU your machine has. If your effect requires a certain type of CPU, you should check this value and return an error indicating that the effect cannot

	run. Adobe After Effects only runs on 68020s and higher, so you don't need to check this field if you just require 68020.
what_fpu	This is set to the return value from Gestalt asking what sort of FPU your machine has. If you require a floating point unit, you should return the PF_OutFlag (see below) indicating such in GLOBAL_SETUP, and then do not execute your floating point code if this value is set to 0 — just do a PF_COPY callback of the input to the output when you get the PF_Cmd_RENDER. See OutFlag description below and sample FPU-requiring code.
qd_globals	This is a pointer to a read only copy of the QuickDraw globals. In Adobe After Effects 1.x, there was a problem and the data in these fields was totally incorrect. This was corrected in After Effects 2.0.
current_time	This is the time of the current frame. It will be set in RENDER. The linear number of the current frame is current_time divided by time_step. All effects sequences start at time 0 for the first frame.
time_step	This is the time difference between frames. This value and current_time and total_time are in units given by time_scale. The time between frames is time_step, not 1. This value will be 0 at SEQUENCE_SETUP if it is not constant for all frames. It will be set correctly in the FRAME calls, even if it's not constant.
total_time	This is the amount of time from the start to the end of the image sequence on which this effect is being invoked. The total number of frames is total_time divided by time_step.
local_time_step	This is the time step in the local composition. local_time_step divided by time_scale will give you the rate duration in the composition in which it is applied.
time_scale	These are the units per second that current_time, time_step, and total_time are in. See QuickTime documentation for an explanation of how these time values work. As a quick example, if time_scale is 30, then the units of current_time, time_step, and total_time are in 30ths of a second. The time_step might then be 3, indicating that the sequence is actually being rendered at 10 frames per second. The total_time might be 105, indicating that the sequence goes for 3.5 seconds.
field	Not implemented.
shutter_angle	This is a value describing the motion blur shutter angle (range is 0 to 1).
width, height	These give the dimensions of the layer. They are not the same as the width and height fields in the input image parameter, which is param[0].

extent_hint This is a rectangle that indicates the intersection of the visible portions of the input and output layers. For an effect that does not do a geometric distortion of the image, copying just this rectangle from the source image to the destination image is sufficient to copy all the image data that the user will see. This can speed up effects very much. If your effect doesn't distort a layer, you can just iterate over only this rectangle of pixels.

output_origin_x, output_origin_y

These specify the origin of the output buffer in the input buffer. They are non-zero only when the effect changes the buffer size.

downsample_x, downsample_y

For speed, the user may have asked for only every Nth vertical or horizontal pixel to be actually rendered by After Effects. The width and height of all effect parameters (including layers) will be automatically adjusted to compensate, but the effect may need to know the downsampling factors to correctly interpret scalar parameters (i.e. sliders) that represent pixel distances in the image. For example, a blur of 4 pixels should be interpreted as a blur of 2 pixels if the down sample factor is 1/2 in each direction. (In After Effects, downsamples are represented as ratios.) This field is set validly in SEQUENCE_SETUP or RESETUP, FRAME_SETUP and FRAME_RENDER.

pixel_aspect_ratio This is a structure describing the pixel aspect ratio (pixel width over pixel height). The NTSC D-1 pixel_aspect_ratio is .9 and PAL D-1 is 1.1.

in_flags These are various flags indicating some Boolean value to the effect module. This is a combination of PF_InFlag values OR-ed together. This field is set for all commands, though most flags make sense only at certain times. Currently there are no defined input flags.

global_data, sequence_data, frame_data

These fields are copied from the values you set in the out data on previous invocations and set here for you to access the shared data as you need it. The fields will only be set if they were allocated during previous commands.

PF_ParamsList Array of Parameter Descriptions

You specify the parameters of your effect when you get the PF_Cmd_PARAMS_SETUP selector by using the PF_ADD_PARAM callback (described below). Before filling in a PF_ParamDef structure with your parameters, you should always zero it out first.

The specific structures of each parameter are defined in AE_Effect.h and examples of each type of parameter are given in the sample code. As an overview, the types of parameters (with short descriptions) are:

PF_Param_LAYER Layer parameters represent movie or image layers in the composition. All effects automatically have 1 layer param, param[0], which is the layer to which they have been

applied. Some effects may have additional layer parameters to do compound effects or multi-channel effects. See the PF_Param_LAYER description in AE_Effect.h for details about how to use additional layer parameters beyond the default parameter, and also see the example code.

PF_Param_SLIDER, PF_Param_FIX_SLIDER

Sliders and Fixed point sliders represent single numerical values. They are the most common type of effect control. You specify a minimum and maximum value, and the user can move a slider or type a number to specify the setting they desire. FIX_SLIDER sliders represent the value as a Fixed (see Inside Mac, Vol II), thus giving you high precision — you specify the number of decimal places of precision you want the user to see. Regular SLIDER sliders represent the value as a simple long. The difference between a Fixed point slider with zero decimal places of precision and a regular slider is that After Effects will still interpolate a Fixed point slider with full Fixed point precision, even though the user only sees the integral part. Your effect would have to ignore the low word of the Fixed point value and use only the integral high word to get the pure integral results. Nonetheless, non-Fixed sliders may well disappear in a future version of After Effects.

PF_Param_ANGLE Angle controls represent an angle in Fixed point degrees (thus, to small fractions of a degree). The angle value is not limited to the 0 to 360 degree range. Users can specify multiple revolution values, if they desire.

PF_Param_CHECKBOX

A checkbox control represents a True or False value. After Effects will only allow a checkbox to be interpolated with None or Hold interpolate behavior (because linear interpolation doesn't make too much sense).

PF_Param_COLOR A color control represents a 24 bit RGB value that the user can choose either with the standard color picker or with an eye dropper tool. In the future, a more sophisticated color control may be added that allows the user to specify an Alpha value for the color as well as RGB values.

PF_Param_POINT A point control represents a point in the image. You specify the default value for the point to After Effects as a value between 0 and 100 in Fixed point with the radix point at bit 16 (i.e. standard Fixed point). Specifying (50,50) in Fixed point indicates the point defaults to the center of the image. The value you are returned for a point control is in absolute pixels with some number of bits of Fixed point accuracy (you specify the location of the radix point). Thus, if you gave (50,50) as the default position and the user applied the effect to a 640 by 480 layer, the default value you would be sent would be (320, 240) in Fixed point. See the example code if you are confused.

PF_Param_POPUP A pop-up represents a list of choices. You specify a list of pop-up entries in standard Mac menu manager string format ("Entry1|Entry2|Entry3" etc.), and we create a pop-up menu.

As with checkboxes, interpolation between pop-up items frequently does not make sense.

Effect Output

The output parameters of an effect are the processed output image, and a variety of flags, error values, and/or potentially allocated Handles that will become input to future invocations. Let's examine the PF_OutData structure, then briefly discuss more details about the PF_LayerDef structure that stores the output (and also the input layer params, which we skimmed over previously).

PF_OutData Structure

The format of the PF_OutData structure is as follows.

```
typedef struct {
   unsigned long           my_version;
   char                    name [PF_MAX_EFFECT_NAME_LEN + 1];
   Handle                  global_data;
   short                   num_params;
   Handle                  sequence_data;
   long                    flat_sdata_size;
   Handle                  frame_data;
   short                   width;
   short                   height;
   Point                   origin;
   PF_OutFlags             out_flags;
   char                    return_msg [PF_MAX_EFFECT_MSG_LEN + 1];
} PF_OutData;
```

You only need to set certain output values for any given PF_Cmd selector. When you can set a field is described along with the usage for that field. Let's look at the fields.

my_version Set this flag to the version of your plug-in code. Use the PF_VERSION macro to construct the proper version value from major, minor, bug etc., versions. After Effects uses this data to decide which effect to load in the case of multiple copies of the same effect.

name This is the name of the effect. PF_STRCPY your value here (see Callbacks below). This field is checked after PF_Cmd_GLOBAL_SETUP.

global_data This is a Handle that you can allocate at PF_Cmd_GLOBAL_SETUP time. It will be passed back to you verbatim in the input parameter block for use later on. The same handle is shared by all instances of the effect. If you have not specified PF_OutFlag_GLOBALS_ARE_CHANGEABLE in the out_flags, this will be written out to disk.

num_params The calling application will sanity check the num_params field vs the times add_param when called (see Callbacks below). For filters, the implicit input layer parameter MUST be included in the parameter count.

sequence_data

This is a Handle that you can allocate at PF_Cmd_SEQUENCE_SETUP time. It will be passed back to you in the input parameter block for later use. This handle is shared across all invocations of this instance of the effect, but not with other sequences. The contents of this handle may be written out to disk. If other handles hang off this block, you must specify the PF_OutFlag_SEQUENCE_DATA_NEEDS_FLATTENING out flag when you get the PF_Cmd_GLOBAL_SETUP command. You will then receive the SEQUENCE_FLATTEN command before your handle is written out. At that time, you should create a flat version of the handle contents, free the old unflat handle, and set this field to the flattened version of the handle. You will receive a PF_Cmd_SEQUENCE_RESETUP call to unflatten this handle (as well as to adjust the sequence data to altered input parameter sizes, etc.). If your sequence data can be flat or unflat, you should store its current state along with the other data, and check that value in Resetup. If the handle is flat, Resetup should unflatten it, free the flat handle, and set this field to the new unflat useable handle.

flat_sdata_size

This field should be the number of bytes of the flattened sequence data handle. Set it in either PF_Cmd_SEQUENCE_SETUP or PF_Cmd_SEQUENCE_FLATTEN. (You will not get the FLATTEN command if you haven't asked for it, so set this in SETUP if you are not flattening data).

frame_data

This is a Handle that you can allocate at PF_Cmd_FRAME_SETUP time. It will be passed to you in the input parameters block as will the global_data and the sequence_data. This will not be written out to disk. There is no longer a particular use for this as After Effects does not interrupt and resume rendering of one frame. Set this field in PF_Cmd_FRAME_SETUP, if you'd like.

width, height, origin

You can set these fields at PF_Cmd_FRAME_SETUP time to indicate that the output image will be larger than the input image. You should set width and height to the size that you want the output buffer to be allocated at. Set origin to the place that the point (0,0) in the input should map to in the new larger output. Thus, if you created a 5 pixel drop shadow up and left, you might set origin to (5, 5).

out_flags

The PF_OutFlag values are used to communicate many things about your effect to the program. You should OR (i.e. use the | operator) multiple values together and set the out_flags field. The values are shown described in the next section.

return_msg

This is a message string (in C string format) that will be interpreted as either an error message or a useful display message (for instance, when handling PF_Cmd_ABOUT). Fill this string with a message you want After Effects to report to the user for you. It will come up in a simple dialog with an OK button. Set the first byte of this string to '\0' to indicate no string -- it is set that way upon entry to your effect. This field is examined after every PF_Cmd.

PF_OutFlags

Here are the values you can use in the out_flags field of the PF_OutData structure.

PF_OutFlag_NONE This is the “empty” setting. Just here for reference.

PF_OutFlag_KEEP_RESOURCE_OPEN

If set, After Effects will keep the plug-ins resource fork open the entire time it is running. This should be set if your plug-in has any of its own resources (not including the required PiPL).

PF_OutFlag_WIDE_TIME_INPUT

Set this flag if the effect calls get_param (see Callbacks below) to inquire a parameter at a time besides the current one (e.g. to get the previous video frame). This must be set, if it’s going to be set, at PF_Cmd_GLOBAL_SETUP.

PF_OutFlag_NON_PARAM_VARY

Set this if the effect uses information other than the parameters in the param list to generate its output at the current time. For instance, if the effect uses the current time of the frame or some random value to decide the output, set this flag. This flag should be sent at PF_Cmd_GLOBAL_SETUP.

PF_OutFlag_SEND_PARAMS_UPDATE

!!! This flag is ignored in After Effects 3.0 !!! Set if the effect wants to update controls values after a parameter has been changed. For instance, if you want a slider to display qualitative text descriptions of the numerical values, you would specify this flag, and then get PF_Cmd_PARAMS_UPDATE whenever the user altered any of the parameters. This flag should be sent at PF_Cmd_GLOBAL_SETUP if it is going to be sent.

PF_OutFlag_SEQUENCE_DATA_NEEDS_FLATTENING

When you allocate a global data handle or a sequence data handle, the app may write the handle out to disk and reuse it later. Pass these flags if the handle is not “flat” (i.e. has pointers or handles hanging off of it). Basically, this gives you a chance to alter the handle contents before it is written out to disk, so you won’t get invalid handles or pointers. Once you have flattened a handle, you will get an opportunity to un-flatten it before the filter needs to continue. For global data, you will be invoked with a PF_Cmd_GLOBAL_SETUP, and the global_data handle in the PF_InData will be non-NULL. You should un-flatten the global_data, free the flat handle memory, and set the global_data handle in the PF_OutData to the unflattened handle. For sequence data, you will be invoked with a PF_Cmd_SEQUENCE_RESETUP call. You should store a Boolean at a common offset in your unflattened and flattened data that says whether the data is flat or not. If you get a PF_Cmd_SEQUENCE_RESETUP and the Boolean indicates the data is flattened, you should unflatten the data, free the flattened data handle, and set the sequence_data handle in the PF_OutData. If you ever set the

data to NULL when you flatten it, you may NOT get the global setup or sequence resetup calls to unflatten it. Instead, you may just get a FRAME_SETUP call with NULL data. Forewarned is forearmed. These flags, indicating if the data will need to be flattened, should be set at PF_Cmd_GLOBAL_SETUP time (yes, even the sequence data flag).

PF_OutFlag_I_DO_DIALOG

Set this if the effect responds to a PF_Cmd_DO_DIALOG, i.e. Does this effect bring up an options dialog box? PF_Cmd_DO_DIALOG is generated when the user presses the Options button on an Effect dialog. This flag should be set at PF_Cmd_GLOBAL_SETUP time.

PF_OutFlag_USE_OUTPUT_EXTENT

The input and output layers are passed with “extent rects” indicating the area of the layer that actually contains visible image data. If the effect changes its behavior based on the extent rect (for instance, by not iterating over the entire image), set these flags so the application will know whether having the extent change should cause the frame to re-render. Specify these flags at PF_Cmd_GLOBAL_SETUP.

PF_OutFlag_SEND_DO_DIALOG

Some filters need their options dialog box to be brought up at least once to be valid. You can set this flag, and the driver app will automatically send a PF_Cmd_DO_DIALOG to the effect when it is applied. The DO_DIALOG will be sent after PF_Cmd_SEQUENCE_SETUP. This flag should be set in PF_Cmd_SEQUENCE_SETUP if it is going to be set.

PF_OutFlag_DISPLAY_ERROR_MESSAGE

Whenever the return_msg field in the PF_OutData is set to a string, After Effects will bring up a simple dialog box containing that string. If you set this flag, the dialog box will be made to look like an error message dialog box. If you don’t set this flag, it will be an undecorated dialog box. Using this flag, an effects module can have and display its own error messages and not worry about the code for dialog boxes — the program will do it for you. Note that this flag can be used to report system errors — set this flag and return an error and After Effects will display the appropriate error message. This way you don’t have to worry about strings for system error messages. This flag can be sent after any command.

PF_OutFlag_I_EXPAND_BUFFER

Starting with After Effects 2.0, effects are able to expand their buffers beyond the current layer's dimensions. This has always been part of the PF specification, but as an extra precaution (and hint to the AE rendering engine) set this flag at PF_Cmd_GLOBAL_SETUP if you plan to expand your buffer.

PF_OutFlag_PIX_INDEPENDENT

Set this flag if the output at a given pixel is not dependent on the values of the pixels around it. If this is set, the pixels After Effects does not care about (because of field rendering, for example) could be filled with garbage colors. Please set this flag at PF_Cmd_GLOBAL_SETUP.

PF_OutFlag_I_WRITE_INPUT_BUFFER

Set this flag if your effect would like to write into the input buffer. This can be useful if you need an scratch buffer, but it also invalidates some speedups in the AE rendering pipeline, so use it with some discretion. Please set this flag at PF_Cmd_GLOBAL_SETUP.

PF_OutFlag_I_SHRINK_BUFFER

Set this flag if you can shrink your buffer based on the extent-rects passed to you in order to be more memory efficient.

PF_OutFlag_WORKS_IN_PLACE

Set this if the plug-in doesn't require separate input and output buffers. If they can share a buffer you'll be able to save some memory.

PF_OutFlag_SQUARE_PIX_ONLY

Not presently used.

PF_OutFlag_CUSTOM_UI

Indicates the plug-in has a custom user interface and want to handle events (so it will receive PF_Cmd_EVENT).

PF_OutFlag_CUSTOM_NTRP

Not presently used.

PF_OutFlag_REFRESH_UI

If you set this flag before exiting your plug-in, After Effects will immediately recall your plug-in with a draw event so you can refresh the user interface.

PF_OutFlag_NOP_RENDER

Set this flag at frame setup time if the current rendering won't effect the image. This allows After Effects to skip some steps (and hence save same time) in its rendering pipeline.

PF_OutFlag_I_USE_SHUTTER_ANGLE

Indicates rendered images depend upon the value of the shutter_angle field.

PF_OutFlag_I_USE_AUDIO

Indicates rendered images depend on sound values (see the three audio callbacks in the next section).

PF_OutFlag_I_AM_OBSOLETE

Set this bit if you want a the plug-in to be available for use, but not to appear in the Effect menu.

PF_LayerDef Structure

Finally, let's look at the structure of the output layer definition. The layer definition structure looks like this.

```
typedef struct PF_LayerDef {
   PF_ParamValue           reserved0;
   PF_ParamValue           reserved1;
   PF_WorldFlags           world_flags;
   PF_Pixel                *data;
   long                    rowbytes;
   long                    width;
   long                    height;
   Rect                    extent_hint;
   long                    platform_ref;
   long                    reserved[8];
   short                   dephault;
} PF_LayerDef;
```

PF_LayerDef structures are frequently called PF_Worlds. The relevant fields are:

world_flags — These flags specify information about the image. Currently, the only supported flag is PF_WorldFlag_WRITEABLE which indicates that you are allowed to alter the image data of the world. This is relevant, because you are usually not allowed to alter the image data of input worlds, but are allowed to alter the data of output. If you check this flag on the input layer, you can sometimes avoid having to allocate scratch image space.

data — This is a pointer to the actual image data. Image data in After Effects is always organized in 32-bit chunky format — that is, sequential long words each contain Alpha, Red, Green, Blue from the high byte to the low byte.

rowbytes — The block of pixels contains 'height' lines each with 'width' pixels followed by some bytes of padding. The 'width' pixels (times four, because each pixel is four bytes long) plus the extra padding adds up to rowbytes bytes. Use this value to move a pointer from scanline to scanline as you traverse the image data.

width, height — These represent the width and height of the image.

extent_hint — For source layers, extent_hint is the smallest rect encompassing all opaque (non-zero alpha) areas of the layer. For output, this encompasses the area that needs to be rendered (i.e. not covered by other layers, needs refreshing, etc.). If your plug-in varies visually based on extent (like a diffusion dither, for example) you should ignore this param and render the full frame each time — effect results should NOT vary based on this field, it only exists to potentially save time.

platform_ref

This field is still unused. To access platform-specific information about a PF_World, use the PF_GET_PLATFORM_REFS macro.

The PF_GET_PLATFORM_REFS macro provides a CGrafPtr and a GDeviceHandle from a PF_World.

Callbacks

Adobe After Effects provides several sets of callbacks: a set for user interaction, a set for ANSI c style string manipulation, a set for math intrinsics, a set of graphics utilities, and a set for color space conversions. There are also some custom user interface callbacks which are described in the next chapter.

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

User Interaction Related Callbacks

Every effect will call some of the interaction callbacks. Function pointers to these callbacks are provided in the PF_InData structure, and macros are defined in AE_Effect.h to give easy access to the routines. The macros to access these functions (along with fake prototypes for the macros and explanations) are listed next.

PF_ADD_PARAM

```
PF_Err PF_ADD_PARAM (
    PF_InData        *in_data,
    PF_ParamIndex    index,
    PF_ParamDefPtr   def );
```

When given the PARAMS_SETUP message, the effect will generally make a series of calls to the add_param routine to define the interface that the After Effects user will see. See the PF_Param types described above, the AE_Effect.h header and the sample code for examples of the use of this callback.

PF_ABORT

```
PF_Err PF_ABORT (PF_InData *in_data);
```

Periodically, you should check if the user wants to interrupt the current processing. The abort proc here will return non-zero if the effects module should suspend its current processing. If you call this routine and it returns a value other than zero, you should return that value when your effect returns. That will let the application know whether the effect completed rendering the output image or not.

PF_PROGRESS

```
PF_Err PF_PROGRESS (
    PF_InData        *in_data,
    long             current,
    long             total );
```

You may wish to display a progress bar while you are processing the image. This routine combines the abort proc user interrupt checking with code that will display a progress bar for you. The current and total params represent a fraction (current divided by total) that describes how far you

are along in your processing. Current should equal total when done. Additionally, this routine will return non-zero if you should suspend/abort your current processing. You should probably try not to call this too frequently (e.g. at every pixel). It is better to call it, say, once per scanline, unless your effect is really, really slow. If total is set to 0, then After Effects will automatically call PF_ABORT.

PF_CHECKOUT_PARAM

```
PF_Err PF_CHECKOUT_PARAM (
    PF_InData        *in_data,
    PF_ParamIndex    index,
    long             what_time,
    long             step
    long             time_scale,
    PF_ParamDef      *param );
```

The checkout_param callback allows you to inquire parameter values at times other than the current one, and allows you to access layer params other than the default input and output layer. See the notes on the “params” structure at the end of AE_Effect.h for details. Another words, it queries the After Effects database so you can look at your param values.

The PF_ParamDef you specify cannot point into the “params” array and the memory must exist elsewhere, such as on the stack. If you checkout a layer parameter and the layer pop-up is currently set to <none>, the returned PF_ParamDef will be filled with zeros. You can check the “data” pointer. If it is NULL, then the layer param is set to <none> and you should do something like faking an all alpha zero layer or some such nonsense. Note that you can inquire layers at other times. Note that params checked out are strictly read only, writing to them could cause unpredictable rendering errors.

PF_CHECKIN_PARAM

```
PF_Err PF_CHECKIN_PARAM (
    PF_InData        *in_data,
    PF_ParamDef      *param );
```

When you have called checkout_param, you must call checkin_param when you are done so After Effects can clean up after itself and you. This is very important for smooth functioning and also to save memory where possible. Once checked in, the fields in the PF_ParamDef will no longer be valid.

PF_REGISTER_UI

```
PF_Err PF_REGISTER_UI (
    PF_InData        *in_data,
    PF_CustomUIInfo  *cust_info );
```

Register a custom user interface element. See the Custom User Interface chapter5 for information.

PF_CHECKOUT_LAYER_AUDIO

```
PF_Err PF_CHECKOUT_LAYER_AUDIO (
    PF_InData        *in_data,
    PF_ParamIndex    index,
    long             start_time,
    long             duration,
    long             time_scale,
    PF_UFixed        rate,
    long             bytes_per_sample,
    long             num_channels,
    long             fmt_signed,
    PF_LayerAudio    *audio );
```

Given an index, start_time, duration, time_scale, rate, bytes_per_sample, num_channels, and fmt_signed, After Effects will return a PF_LayerAudio structure. This is an abstract chunk of data which can be feed into PF_GET_AUDIO_DATA to extract some meaningful information. As with PF_CHECKOUT_PARAM, this data should be considered strictly read only.

PF_CHECKIN_LAYER_AUDIO

```
PF_Err PF_CHECKIN_LAYER_AUDIO (
    PF_InData        *in_data,
    PF_LayerAudio    audio );
```

After calling PF_CHECKOUT_LAYER_AUDIO, you must call this to check the layer audio back in after examining it.

PF_GET_AUDIO_DATA

```
PF_Err PF_GET_AUDIO_DATA (
    PF_InData        *in_data,
    PF_LayerAudio    audio,
    PF_SndSamplePtr  *data0,
    long             *num_samples0,
    PF_UFixed        *rate0,
    long             *bytes_per_sample0,
    long             *num_channels0,
    long             *fmt_signed0 );
```

All the parameters after audio are optional. Given a PF_LayerAudio structure, this returns all the indicated information.

Kernel Flags

Many functions work with "kernels" or matrices of values. These matrices can be of different types, of different arrangements, and can be generated or treated in different ways. The KernelFlags are used in a variety of functions to determine how the matrices should be created and used. You should OR together any flags you need. Which flags are relevant for a given routine are documented along with the prototype for the routines below.

For each row below you can choose one of the entries. The first entry is always the default and has value 0.

Kernel Flags	Description
PF_KernelFlag_2D PF_KernelFlag_1D	Use 1D for a one dimensional kernel or 2D for a two dimensional kernel.
PF_KernelFlag_UNNORMALIZED PF_KernelFlag_NORMALIZED	Use the NORMALIZED flag to equalize the kernel, forcing the volume under the kernel surface to be the same as the volume under the covered area of pixels. Otherwise, it will be unnormalized.
PF_KernelFlag_CLAMP PF_KernelFlag_NO_CLAMP	Use the CLAMP flag to force values to be clamped into their valid range (that is determined by the type of item (char, fixed, long).
PF_KernelFlag_USE_LONG PF_KernelFlag_USE_CHAR PF_KernelFlag_USE_FIXED PF_KernelFlag_USE_UNDEFINED	Use USE_LONG to treat the kernel as an array of longs valued from 0 to 255, use USE_CHAR to treat the kernel as an array of unsigned chars from 0 to 255, and use USE_FIXED to treat the kernel as an array of fixeds from 0 to 1. Note: At present USE_LONG is the only one of these flags which is implemented!
PF_KernelFlag_HORIZONTAL PF_KernelFlag_VERTICAL	Use the HORIZONTAL flag to apply a 1D convolution horizontally, or VERTICAL to apply it vertically
PF_KernelFlag_TRANSPARENT_BORDERS PF_KernelFlag_REPLICATE_BORDERS	Use REPLICATE_BORDERS to replicate border pixels when sampling off the edge, use TRANSPARENT_BORDERS to treat pixels off the edge as alpha zero (black). Note: At present the REPLICATE_BORDERS flag is not implemented and will be ignored!
PF_KernelFlag_STRAIGHT_CONVOLVE PF_KernelFlag_ALPHA_WEIGHT_CONVOLVE	Use STRAIGHT_CONVOLVE to indicate straight convolution, use ALPHA_WEIGHT_CONVOLVE to tell the convolution code to alpha-weight the contributions of pixels to the resulting convolved output. Note: At present the ALPHA_WEIGHT_CONVOLVE flag is not implemented and will be ignored.

Graphics Utility Callbacks

The graphics utility callbacks are extensively documented in the AE_EffectCB.h file. An overview of the callbacks available and their capabilities will be given here, but you should look at the header file and the example code for details on actually using the utilities.

Plug-ins do not have to use the graphics utility callbacks, but there are advantages in doing so. First, it will frequently be easier to use these callbacks rather than re-implementing the wheel (so to speak). The callbacks will, in general, be relatively fast and will work correctly. Second, on machines that have a DSP (Digital Signal Processor) or other hardware accelerator installed, some of these callbacks may be accelerated automatically, transparent to the plug-in. In addition, machines equipped with multiple processors will be able to divide the processing amongst the processors. Third, it will make it easier to port your plug-in code to another platform.

Accessing the utility callbacks can be a little complex, because the pointer to the structure of function pointers in the PF_InData is defined as a void * in

AE_Effect.h. Macros are defined in AE_EffectCB.h to make it easier to use these utility functions — the prototypes given for the functions here are the prototypes for the macros (which are not exactly the same as the real prototypes). See the Macros section of AE_EffectCB.h for further explanation.

PF_SUBPIXEL_SAMPLE

```
PF_Err PF_SUBPIXEL_SAMPLE (
    PF_Fixed         x,
    PF_Fixed         y,
    const PF_SampPB  *params,
    PF_Pixel         *dst_pixel );
```

Use this to inquire the appropriate alpha weighted interpolation of colors at a non-integral point in a source image, in high quality. Nearest neighbor sampling is used in low quality.

Because the sampling routine, if used, will typically be called many times, it is convenient to copy the function pointer out of the callbacks structure and into a register or onto the stack to speed up your inner loop. See the sample code for an example.

PF_AREA_SAMPLE

```
PF_Err PF_AREA_SAMPLE (
    PF_Fixed         x,
    PF_Fixed         y,
    const PF_SampPB  *params,
    PF_Pixel         *dst_pixel );
```

Use this to calculate the appropriate alpha weighted average of an axis-aligned non-integral rectangle of color in a source image, in high quality. Nearest neighbor in low quality. This routine will be vectored to the proper alpha-version. Because of overflow issues, this can only average a maximum of a 256 pixel by 256 pixel area (i.e. x and y range < 128 pixels). In After Effects 3.x this callback uses a box filter for pixel averaging.

PF_BLEND

```
PF_Err PF_BLEND (
    const PF_World   *src1,
    const PF_World   *src2,
    PF_Fixed         ratio,
    PF_World         *dst );
```

This blends two images with one another. This does an alpha-weighted mixture of the colors. This is provided because ALL effects should have a state in which there is no visual change to the source image. This can often be realized by providing a “blend-with-source” slider. It is possible that this will have different high and low qual versions.

PF_CONVOLVE

```
PF_Err PF_CONVOLVE (
    PF_World         *src,
    const Rect       *area,
    PF_KernelFlags   flags,
    short            kernel_size,
    void             *a_kernel,
    void             *r_kernel,
    void             *g_kernel,
    void             *b_kernel,
    PF_World         *dst );
```

Convolve an image with an arbitrary size kernel on each of the a, r, g, and b channels separately. You can specify a rectangle to convolve (for instance, the extent_hint), or pass NULL to convolve the entire image. Using the kernel flags you can specify a full two-dimensional convolution or a one-dimensional horizontal or vertical convolution, as well as kernel normalization, range clamping, edge behavior, and alpha-weighting. This callback may have different high and low quality versions.

This looks for the following kernel flags (see the defines in AE_EffectCB.h):

Use 1D or 2D
Use Clamp or No Clamp
Use long, char, and fixed
Use straight or alpha-weighted convolve
Plus if 1D is specified, use horizontal or vertical

PF_COPY

```
PF_Err PF_COPY (
    PF_World         *src,
    PF_World         *dst,
    Rect             *src_r,
    Rect             *dst_r );
```

This blits a region from one PF_World to another. This is an alpha-preserving (unlike CopyBits), 32-bit only, antialiased stretch blit. This routine will be vectored based on the current alpha channel preference.

PF_FILL

```
PF_Err PF_FILL (
    const PF_Pixel   *color,
    const Rect       *dst_rect,
    PF_World         *world );
```

This fills a rectangle in the image with the given color. Setting the color pointer to NULL will fill the rectangle with black (and alpha zero). Setting the rectangle to NULL fills the entire image. If you use this routine to fill with a transparent color, be sure to check your current alpha mode to avoid filling a rectangle with illegal color values. (With premultiplied alpha, r, g, b must not be greater than a.)

PF_GAUSSIAN_KERNEL

```
PF_Err PF_GAUSSIAN_KERNEL (
    double           kRadius,
    PF_KernelFlags   flags,
    double           multiplier,
    short            *diameter,
    void             *kernel );
```

Generate a kernel with a Gaussian distribution of values. Using the kernel flags you can specify a one or two

dimensional array of values and a normalized or unnormalized distribution. This callback will be the same for high and low quality.

This looks for the following kernel flags (see the defines in AE_EffectCB.h):

Use 1D or 2D
Use Normalized or Unnormalized
Use longs, chars, and fixed

The multiplier parameter value is multiplied by every value generated. In general you should pass 1.0, but this lets you adjust the "fuzziness" of the kernel.

The diameter parameter is the actual integral width of generated kernel. This will always currently be (int)ceil(kRadius) * 2 + 1. You need to know this because the "kernel" array must be already allocated

Upon entry to this routine the kernel parameter is a diameter by diameter array of values allocated by you, of longs, chars, or fixeds. It points to the kernel upper left corner.

PF_ITERATE

```
PF_Err PF_ITERATE (
    long              progress_base,
    long              progress_final,
    PF_World          *src,
    const Rect        *area,
    long              refcon,
    PF_Err            (*pix_fn)(
                           long refcon,
                           long x,
                           long y,
                           PF_Pixel *in,
                           PF_Pixel *out ),
    PF_World          *dst );
```

This invokes a function you specify on a region of pixels in the source and dest images. You give a refcon, and the function is invoked with that refcon, plus the x and y coordinates of the current pixel, plus pointer to that pixel in the source and destination images. You can specify a rectangle to iterate over (for instance, the extent_hint), or pass NULL for the rect param to iterate over every pixel where the worlds overlap. If you pass the src world as NULL, this will just iterate over the dst. This function is quality independent.

You should not depend upon the pixels being traversed in any particular order and you should consider all the parameters (except dst) to be read-only while After Effects is processing. The reason for this is so the image can be split up by After Effects for processing on multiple processor machines.

This callback automatically includes progress and abort checking, so you don't need to include that in your pixel function.

PF_ITERATE_ORIGIN

```
PF_Err PF_ITERATE_ORIGIN (
    long            progress_base,
    long            progress_final,
    PF_World        *src,
    const Rect      *area,
    const Point     *origin,
    long            refcon,
    PF_Err          (*pix_fn)(
                        long refcon,
                        long x,
                        long y,
                        PF_Pixel *in,
                        PF_Pixel *out ),
    PF_World        *dst );
```

This routine is similar to PF_ITERATE except that it lets you specify an offset from the input into the output. For example, if your output buffer is smaller than your input buffer, pass (in_data->output_origin_x, in_data->output_origin_y) as the origin parameter, and NULL as area, and PF_ITERATE_ORIGIN will offset the src pixel pointer appropriately to your pix_fn.

PF_ITERATE_LUT

```
PF_Err PF_ITERATE_LUT (
    PF_InData       *in_data,
    long            progress_base,
    long            progress_final,
    PF_World        *src,
    const Rect      *area,
    unsigned char   *a_lut0,
    unsigned char   *r_lut0,
    unsigned char   *g_lut0,
    unsigned char   *b_lut0,
    PF_World        *dst );
```

Iterate Look Up Table. This is the same as PF_ITERATE except it uses a look up table instead of a function pointer. The look up table values for alpha, red, green, blue are specified using the *_lut0 parameters. If any of these are 0, then the identity look up table will be used.

PF_TRANSFER_RECT

```
PF_Err PF_TRANSFER_RECT (
    PF_InData               *in_data,
    PF_Quality              quality,
    PF_ModeFlags            m_flags,
    PF_Field                field,
    const Rect              *src_rec,
    const PF_World          *src_world,
    const PF_CompositeMode  *comp_mode,
    const PF_MaskWorld      *mask_world0,
    long                    dest_x,
    long                    dest_y,
    PF_World                *dst_world );
```

This performs a rect to rect blend using any of the supported After Effects transfer modes, with an optional mask.

PF_TRANSFORM_WORLD

```
PF_Err PF_TRANSFORM_WORLD (
    PF_InData               *in_data,
    PF_Quality              quality,
    PF_ModeFlags            m_flags,
    PF_Field                field,
    const PF_World          *src_world,
    const PF_CompositeMode  *comp_mode,
    const PF_MaskWorld      *mask_world0,
    const PF_FloatMatrix    *matrices,
    long                    num_matrices,
    Boolean                 src2dst_matrix,
    const Rect              *dest_rect,
    PF_World                *dst_world );
```

This callback signifies the heart of the After Effects rendering engine. Given a PF_World (*src_world) and a matrix, or an array of matrices, this transforms and blends using any of the supported After Effects transfer modes with an optional mask. The matricies pointer points to a matrix array used for motion-blur.

PF_PREMUL

```
PF_Err PF_PREMUL (
    Boolean         forward,
    PF_World        *dst );
```

Converts to and from having r, g, and b color values premultiplied with black to represent the alpha channel. High qual same as low qual.

PF_PREMUL_COLOR

```
PF_Err PF_PREMUL_COLOR (
    PF_World         *src,
    PF_Pixel         *color,
    Boolean          forward,
    PF_World         *dst );
```

To convert to and from having r, g, and b color values premultiplied with any color to represent the alpha channel. High qual same as low qual.

PF_NEW_WORLD

```
PF_Err PF_NEW_WORLD (
    short            width,
    short            height,
    Boolean          blank,
    PF_World         *world );
```

This creates a new PF_World from scratch for you. You must dispose of it. This is quality independent.

PF_DISPOSE_WORLD

```
PF_Err PF_DISPOSE_WORLD (PF_World *world);
```

This disposes a PF_World, deallocating pixels, etc. Only call it on worlds you have created. Quality independent.

GET_PLATFORM_REFS

```
PF_Err GET_PLATFORM_REFS (
    PF_InData       *in_data,
    PF_World        *world,
    void            **plat_1,
    void            **plat_2 );
```

This routine will return two platform-specific long words for a given PF_World. In the case of the Macintosh, the first will be a CGrafPtr and the second will be a GDeviceHandle.

get_callback_addr

```
PF_Err (*get_callback_addr) (
    PF_ProgPtr      effect_ref,
    PF_Quality      quality,
    PF_ModeFlags    mode_flags,
    PF_CallbackID   which_callback,
    PF_CallbackFunc *fn_ptr );
```

Chances are good that you will never use this callback. No macro is provided for easy access. This is the callback to get addresses of callback functions at different qualities or alpha modes. See the large comment in the Callback Selectors section of AE_EffectCB.h. You would use this to circumvent the nearest neighbor behavior of the sampling functions at low quality, if you really needed to.

Intrinsic Callbacks

Along with the variety of graphics utilities, we also provide a block of ANSI standard routines so that plug-ins will not need to include the ANSI library to use standard functions. We give function pointers to a large number of math functions (trig functions, sqrt, logs, etc.). Macros are defined in AE_EffectCB.h to access these functions. In general, the macro looks just like the ANSI function only with a PF_ prefix and in all caps; for example, to call sin(x) you would write PF_SIN(x).

Here are the ANSI routines that can be called through After Effects. These all return a double. All angles are expressed in radians, use PF_RAD_PER_DEGREE to convert from degrees to radians if necessary.

PF_ACOS

```
r PF_ACOS (double x);
```

This returns the arc cosine of x.

PF_ASIN

```
r PF_ASIN (double x);
```

This returns the arc sine of x.

PF_ATAN

```
r PF_ATAN (double x);
```

This returns the arc tangent of x.

PF_ATAN2

```
r PF_ATAN2 (double x, double y)
```

This returns atan(y/x)

PF_CEIL

```
r PF_CEIL (double x);
```

This returns the next integer above x.

PF_COS

```
r PF_COS (double x);
```

This returns the cosine of x.

PF_EXP

```
r PF_EXP (double x);
```

This returns e to the power of x.

PF_FABS

```
r PF_FABS (double x);
```

This returns the absolute value of x.

PF_FLOOR

```
r PF_FLOOR (double x);
```

This returns the closest integer below x.

PF_FMOD

```
r PF_MOD (double x, double y);
```

This returns x modulus y.

PF_HYPOT

```
r PF_HYPOT (double x, double y);
```

This returns the hypotenuse of x and y which is sqrt(x*x + y*y).

PF_LOG

```
r PF_LOG (double x);
```

This returns the natural log (ln) of x.

PF_LOG10

```
r PF_LOG10 (double x);
```

This returns the log (base 10) of x.

PF_POW

```
r PF_POW (double x, double y);
```

This returns x to the power of y.

PF_SIN

```
r PF_SIN (double x);
```

This returns the sine of x;

PF_SQRT

```
r PF_SQRT (double x);
```

This returns the square root of x.

PF_TAN

```
r PF_TAN (double x);
```

This returns the tangent of x.

ANSI Callbacks

Here are a couple more useful ANSI c style callbacks.

PF_SPRINTF

```
PF_Err PF_SPRINTF (
    char        *s1,
    const char  *s2,
    ... );
```

This emulates the c sprintf routine.

PF_STRCPY

```
PF_Err PF_STRCPY (
    char        *s1,
    const char  *s2,
    ... );
```

This emulates the c strcpy routine.

Colorspace Conversion Callbacks

The following callbacks provide functions for converting between various colorspaces. The callbacks reference the following small structures:

This describes an RGB (Alpha, Red, Green, Blue) pixel:

```
typedef struct {
    unsigned char   alpha, red, green, blue;
} PF_Pixel;
```

This describes an HLS (Hue, Lightness, Saturation) pixel:

```
typedef fixed PF_HLS_PIXEL[3]
```

This describes a YIQ (luminance, inphase chrominance, quadrature chrominance) pixel:

```
typedef fixed PF_YIO_PIXEL[3]
```

RGB_TO_HLS

```
PF_Err PF_RGB_TO_HLS (
    PF_Pixel       *rgb,
    PF_HLS_Pixel   hls );
```

Given an RGB pixel, this returns an HLS (hue, lightness, saturation) pixel. HLS values are scaled from 0 to 1 in fixed point.

PF_HLS_TO_RGB

```
PF_Err PF_HLS_TO_RGB (
    PF_HLS_Pixel   hls,
    PF_Pixel       *rgb );
```

Given an HLS pixel, this returns an RGB pixel.

PF_RGB_TO_YIQ

```
PF_Err PF_RGB_TO_YIQ (
    PF_Pixel       *rgb,
    PF_YIQ_Pixel   yiq );
```

Given an RGB pixel, this returns a YIQ (luminance, inphase chrominance, quadrature chrominance) pixel. Y is 0 to 1 in fixed point, I is -0.5959 to 0.5959 in fixed point, and Q is -0.5227 to 0.5227 in fixed point.

PF_YIQ_TO_RGB

```
PF_Err PF_YIQ_TO_RGB (
    PF_HLS_Pixel   yiq,
    PF_Pixel       *rgb );
```

Given a YIQ pixel, this returns an RGB pixel.

PF_LUMINANCE

```
PF_Err PF_LUMINANCE (
    PF_Pixel       *rgb,
    long           *lum100 );
```

Given an RGB pixel, this returns 100 times its luminance value (0 to 25500).

PF_HUE

```
PF_Err PF_HUE (
    PF_Pixel       *rgb,
    long           *hue );
```

Given an RGB pixel, this returns its hue angle mapped from 0 to 255, where 0 is 0 degrees and 255 is 360 degrees.

PF_LIGHTNESS

```
PF_Err PF_LIGHTNESS (
    PF_Pixel       *rgb,
    long           *lightness );
```

Given an RGB pixel, this returns its lightness value (0 to 255).

PF_SATURATION

```
PF_Err PF_SATURATION (
    PF_Pixel       *rgb,
    long           *saturation );
```

Given an RGB pixel, this returns its saturation value (0 to 255).

5 Custom User Interface

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

There are 2 example plug-ins, one which uses a custom UI in the composition window and one which uses a custom UI in the effects window, to demonstrate the new custom UI API. While reading the descriptions below, also refer to the header file, AE_EffectUI.h.

In reading the descriptions below, note the term “context” signifies the environment in which you will get events. Generally this corresponds with a window.

Getting UI Events

You can inform After Effects that you wish to utilize a custom user interface by setting the PF_OutFlag_CUSTOM_UI flag in the out_flags field of the PF_OutData structure (see PF_OutData in the Effects Filters chapter) when responding to a PF_Cmd_GLOBAL_SETUP.

After getting a PF_Cmd_PARAMS_SETUP you can indicate what UI events you wish to receive by filling in the PF_CustomUIInfo structure and calling PF_REGISTER_UI.

Upon receiving the PF_Cmd_EVENT command selector mentioned in chapter 2, the *extra parameter will point to the following PF_EventExtra structure which describes the UI event.

```
typedef struct {
    PF_ContextH                 contextH;
    PF_EventType                e_type;
    PF_EventUnion               u;
    PF_EffectWindowInfo         effect_win;
    PF_EventCallbacks           cbs;
    PF_EventInFlags             evt_in_flags;
    PF_EventOutFlags            evt_out_flags;
} PF_EventExtra;
```

contextH Context Handle. This is a handle to the context which can be drawn in. The PF_Context structure is listed below. There are 4 long words in the structure which the plug-in can use for any specific data it wishes to retain. These can be pointers, handles, or simply long words.

e_type Event Type. This is the event type which identifies the kind of UI event to respond to. The event types are listed below.

u — Event Union. This is a structure containing information specific to the event that has taken place. See PF_EventUnion below for the structure definition.

effect_win — Effect Window Info. This contains additional info about the event if it's taking place in the effects window. See PF_EffectWindowInfo structure below for additional information.

cbs — Event Callbacks. This is a callback structure which provides the addresses of callbacks you can use in your UI code. Most of the callbacks provide coordinate mapping routines. They are listed below.

evt_in_flags — Event In Flags. This currently contains only one value, named PF_EI_DONT_DRAW, which you should examine before drawing into a comp or layer window. If this flag is set you should avoid drawing.

evt_out_flags — Event Out Flags. This currently only contains one value, named PF_EO_HANDLED_EVENT, which you can set. Setting it indicates to After Effects that you've handled an event which shouldn't be further propagated.

PF_Context Structure

The PF_Context structure is returned in the PF_EventExtra structure and is used to identify the context (or window) where the UI event is happening. The structure looks like this:

```
typedef struct {
    unsigned long   magic;
    PF_WindowType   w_type;
    void            *cgrafptr;
    long            reserved_flt;
    long            plugin_state[4];
} PF_Context;
```

magic — This is used internally for integrity checking and should not be altered.

w_type — This is a PF_WindowType which identifies the type of context window. The possible values are:

Window Type	Name	Value
Composition	PF_Window_COMP	0
Layer	PF_Window_LAYER	1
Effect	PF_Window_EFFECT	2
Preview	PF_Window_PREVIEW	3

cgrafptr — This is the Macintosh cgrafptr for the current context.

reserved_flt — Reserved, please do not disturb.

plugin_state — An array of 4 longs which the plug-in can use to store state information specific to a given context.

Event Types (PF_EventType)

The event type identifies the kind of UI event that has just taken place. Depending on the kind of event, the PF_EventUnion parameter will indicate more information about the event. The event types are:

PF_Event_NEW_CONTEXT

The user creates a new context, probably by opening a window for events. The plug-in is allowed to store state information inside the context using the context handle. PF_EventUnion will be empty.

PF_Event_ACTIVATE The user has activated a new context, probably by bringing a window into the foreground. PF_EventUnion will be empty.

PF_Event_DO_CLICK This is a Click Event, the PF_EventUnion parameter will contain a PF_DoClickEventInfo structure (listed below). The plug-in must handle a mouse click inside of a context, but may block until a mouseup, if desired.

PF_Event_DRAG This is also a Click Event, the PF_EventUnion parameter will contain a PF_DoClickEventInfo structure (listed below). The plug-in can request this event by returning from a do_click with a send_drag. This can be used in the middle of a drag operation so After Effects can see the data the user has changed and can update the standard user interface elements. This is demonstrated by the custom UI in fx window plug-in sample.

PF_Event_DRAW This is a Draw Event, the PF_EventUnion parameter will contain a PF_DrawEventInfo structure (listed below).

PF_Event_DEACTIVATE

The user has deactivated a context, perhaps by bringing another window into the foreground. PF_EventUnion will be empty.

PF_Event_CLOSE_CONTEXT

A context is closed by the user. PF_EventUnion will be empty.

PF_Event_IDLE A context is open but nothing much is happening right now. PF_EventUnion will be empty.

PF_Event_KEYDOWN

This is a keyboard event, the PF_EventUnion parameter will contain a PF_KeyDownEvent structure (listed below).

Event Unions (PF_EventUnion)

The event type (listed above) identifies the kind of event. There are basically 3 kinds: Click Event, Draw Event, and Key Down Event. The u parameter will contain one of the following structures, depending upon the kind of event.

Click Event

A Click Event return the following structure. You'll get this when there is any kind of mouse click or drag.

```
typedef struct {
    unsigned long   when;
    Point           screen_point;
    long            num_clicks;
    long            modifiers;
    long            continue_refcon[4];
    Boolean         send_drag;
    Boolean         last_time;
} PF_DoClickEventInfo;
```

when — Indicates the time at which the click occurred. It is the value of TickCount().

screen_point — Screen coordinate where the click occurred.

num_clicks — The number of mouse clicks that occurred.

modifiers — What, if any, modifier keys were held down when the click happened.

continue_refcon — An array of 4 longs that the plug-in can use to store information during a click-drag-drag sequence.

send_drag — When you get a click event, if you want to then drag the mouse, set this flag. The next click event will then effectively be a drag event.

last_time — This will be set when the drag event ends, another words the user has released the mouse button.

Draw Event

A Draw Event returns the following structure. You'll get this when there's any kind of draw event.

```
typedef struct {
    Rect            update_rect;
    long            depth;
    void            *gdeviceH;
} PF_DrawEventInfo;
```

update_rect — This is the rect, in the context windows coordinate system, which is being draw in. Callbacks are provided, see below, for converting between the various coordinate systems used by the various After Effects windows.

depth — This is the pixel depth of the device to draw into.

***gdeviceH** Pointer to the Macintosh gdevice handle.

Key Down Event

A Key Down Event returns the following structure. You'll get this whenever the user presses a key.

```
typedef struct {
    unsigned long   when;
    Point           screen_point;
    long            char_code;
    long            key_code;
    long            modifiers;
} PF_KeyDownEvent;
```

when Indicates the time at which the click occurred. It is the value of TickCount().

screen_point Screen coordinate of the mouse pointer when the key was pressed.

char_code Character code corresponding to the key pressed.

key_code Key code of the key pressed.

modifiers What, if any, modifier keys were held down during the mouse click.

Effect Window Information (PF_EffectWindowInfo)

If an event takes place in an effects window, the following structure will be sent in the PF_EffectWindowInfo field.

```
typedef struct {
    PF_ParamIndex   index;
    PF_EffectArea   area;
    Rect            current_frame;
    Rect            param_title_frame;
    long            horiz_offset;
} PF_EffectWindowInfo;
```

index This indicates which control in the effect window is being affected. The controls are numbered from 0 to the number of controls minus 1.

area This indicates if the control title or the control itself are begin affected.

Area	Name	Value
Title	PF_EA_PARAM_TITLE	1
Control	PF_EA_CONTROL	2

current_frame A rect indicating the full frame of the area occupied by the control.

param_title_frame A rect indicating the title area of the control.

horiz_offset A horizontal offset from the left side of the title area in which to draw into the title.

UI Callbacks (PF_EventCallbacks)

After Effects provides the following UI event callbacks. These are mostly used for transposing coordinate systems. Similar to the effects callbacks, function pointers to these callbacks are provided in the PF_EventCallbacks structure and macros are defined in AE_EffectUI.h to give easy access to the routines. The macros to access these functions (along with fake prototypes for the macros) and explanations are listed next.

Please note these macros default a few parameters for simplicity, in particular the refcon and context handle. Like the effects callbacks, the refcon assumes you have a local named "extra". The default context is the current context. You can see where these default parameters are defined by looking at the PF_EventCallbacks structure in AE_EffectUI.h. You can override the defaults by either accessing the callbacks through the PF_EventExtra structure or by modifying the macros which are located at the bottom of the header file.

LAYER_TO_COMP

```
PF_Err LAYER_TO_COMP (
    long            curr_time,
    long            time_scale,
    PF_FixedPoint   *pt );
```

This transforms layer window coordinates to the composition window coordinates.

COMP_TO_LAYER

```
PF_Err COMP_TO_LAYER (
    long            curr_time,
    long            time_scale,
    PF_FixedPoint   *pt );
```

This transforms composition window coordinates to the layer window coordinates.

GET_COMP2LAYER_XFORM

```
PF_Err GET_COMP2LAYER_XFORM (
    long            curr_time,
    long            time_scale,
    long            *exists,
    PF_FloatMatrix  *comp2layer );
```

This returns the transformation matrix used to convert from the composition window to the layer window.

GET_LAYER2COMP_XFORM

```
PF_Err GET_LAYER2COMP_XFORM (
    long            curr_time,
    long            time_scale,
    PF_FloatMatrix  *layer2comp );
```

This returns the transformation matrix used to convert from the layer window to the composition window.

SOURCE_TO_FRAME

```
PF_Err SOURCE_TO_FRAME (PF_FixedPoint *pt);
```

This transforms the source coordinates identified by *pt of the current context to screen coordinates.

FRAME_TO_SOURCE

```
PF_Err FRAME_TO_SOURCE (PF_FixedPoint *pt);
```

This transforms the screen coordinates identified by *pt to the source coordinates of the current context.

INFO_DRAW_COLOR

```
PF_Err INFO_DRAW_COLOR (PF_Pixel color);
```

This performs the standard color information reporting into the info window which happens as the user moves the mouse over the composition window.

INFO_DRAW_TEXT

```
PF_Err INFO_DRAW_TEXT (
    char            *text1,
    char            *text2 );
```

This displays the given text in the info window when an object is selected in the comp window. text1 is the first line and text2 is the second line in the info window.

INFO_GET_PORT

```
PF_Err INFO_GET_PORT (void **cgrafptr_addr);
```

This returns the Macintosh grafptr for the info window so you can draw whatever you want into it.

6 Input & Output Plug-Ins

After Effects 2.0 added, and 3.1 carries forward, support for input and output plug-ins. Specifically, After Effects supports the Adobe Photoshop File Format Module (hereafter: 8BIF) interface for still images and sequences of still images. While the 8BIF specification is fine for still images, it does not easily accommodate all the input/output options a user would want in the After Effects environment. For example, time-based multi-frame file formats and input/output formats which do not correspond directly with the file system are difficult to implement with the 8BIF specification.

Starting with version 2.0.2, After Effects supports an extension of the 8BIF specification for time-based and non-file based formats, hereafter referred to as the FXIF specification. For example, one could write an FXIF plug-in to talk directly to digital disk recorders or to support a file format with multiple frame samples in one file.

This chapter describes the FXIF extension to the Adobe's 8BIF specification. It therefore assumes that the reader is already familiar with creating plug-ins for Photoshop, specifically File Format Modules. If you are not, see "Image Format Module Interface for Adobe Photoshop," part of the Adobe Photoshop 3.0 Plug-In Software Development Toolkit.

Please note that the 8BIF specification is different from Photoshop's Acquire/Export specification, though they are similar enough that if you have an Acquire or Export plug-in, the conversion should be relatively painless.

Resource Structures

'PiPL' Resource Structure

Every FXIF module must have a 'PiPL' resource. It is identical in structure and function to that of an 8BIF module. It should have the same resource ID as the 'FXMF' code resource and the 'FXMF' module flag resource.

'FXMF' PiPL Atom

The 'FXMF' PiPL atom communicates information about the module that is not described by the standard Photoshop file format specification. For example, a plug-in uses this resource to tell the host whether or not the format directly corresponds to the file system.

FXMF PiPL Atom

Major version (2 bytes)	The major version number of the plug-in specification. This document describes version 1 of the specification.
Minor version (2 bytes)	This minor version number of the plug-in specification.
Module flags (4 bytes)	Flags indicating to the host additional details about the nature of the format. All bits which are not specified must be set to zero. Note that some of these flags are redundant with information in the PiPL. Wherever the two differ, the Module Flags will be used. Please see the header file and the PiPL template for the definition of the module flags.
Reserved (4 bytes)	Reserved for future use. Set to zero.

Interface Record Structure

The stuff parameter contains a pointer to a FormatRecord structure, which is identical to the structure defined in the 8BIF specification. See “Image Format Module Interface for Adobe Photoshop” for details on each field in the structure.

Information added for the FXIF specification can be found above the stuff pointer in memory. The four bytes at (stuff - 4) contain a TimeInfoHandle, and the four bytes at (stuff - 8) describe the operation (selector) the plug-in is being asked to perform. This selector is only valid when the selector parameter in the main entry point is set to 0.

The time-based information is placed above the stuff pointer so you may safely ignore it in any code that you are converting from 8BIF modules. You may wish to declare a FormatRecordTPtr called stuffT on the stack and add the line

```
    stuffT = (stuff - sizeof(TimeExtension))
```

You can then use the stuffT pointer just like the stuff pointer, except you now have easier access to the time-extension information.

Time Extension Structures

Here is the TimeExtension structure mentioned above.

```
typedef struct {
    long            time_selector;
    TimeInfoHandle  time_info;
} TimeExtension;
```

time_selector This value is the FXIF plug-in selector. When the selector parameter to the plug-in's entry point is 0, check this value for the FXIF selector value. Otherwise, use the regular selector value

time_info The time info structure is described next.

```
typedef struct {
    long            time_info_version;
    char            error_msg [PI_FORMAT_T_MESSAGE_LEN+1];
    PITimeRecord    duration;
    PITimeRecord    poster_time;
    Fixed           fixed_fps;
    char            read_name [PI_FORMAT_T_NAME_LEN+1];
    char            read_message [PI_FORMAT_T_MESSAGE_LEN+1];
    long            start_smpte_frames;
    short           time_base;
    char            reel_name[16];
    PITimeRecord    read_time;
    PITimeRecord    read_dur;
    Rect            read_rect;
    char            write_name [PI_FORMAT_T_NAME_LEN+1];
    char            write_message [PI_FORMAT_T_MESSAGE_LEN+1];
    Fixed           write_fixed_fps;
    PITimeRecord    write_duration;
    long            frame_num_to_add;
    long            frames_to_add;
    Boolean         was_compressed;
    long            origin_h;
    long            origin_v;
    PIAlphaLabel    alpha_label;
    FIEL_Label      interlace_label;
} TimeInfo, *TimeInfoPtr, **TimeInfoHandle;
```

The TimeInfo handle is used to communicate time information between the plug-in and the host. It is primarily used as a communication mechanism. In After Effects, the contents of the TimeInfo handle are not saved to disk as part of the project file and need to be recreated by the plug-in each time the saved project is opened. For a non-file module though, you want to be able to save information that might have required lengthy user-interaction. Thankfully, the standard 8BIF specification provides a way to do this via the revertInfo field in the FormatRecord. You may therefore want to store a duplicate of the TimeInfo handle in your private revertInfo. (Note, however, that revertInfo must be a flat structure, i.e. there should be no pointers or handles hanging off of it.)

time_info_version — The high word is the major version and the low word is the sub-version.

error_msg — Set this to a non-null string with *result is an error code. This will cause the host to bring up an alert dialog.

duration — Set this field to the length of the input sequence. When the plug-in is first called for a given input sequence, this value will be invalid (duration.value and duration.scale will be 0.) For each read thereafter, it will be set to the value you first set it to. Set duration in the Read calling sequence.

poster_time — Set this field to the time you wish to be the “thumbnail” time for the input sequence. When the plug-in is first called for a given input sequence, this value will be invalid (duration.value and duration.scale will be 0.) Set poster_time in the Read calling sequence.

fixed_fps — Set this field to the (fixed-point) frame-rate of the input sequence during the Read calling sequence.

read_name — Fill this field with the name of your input sequence during the Read sequence. This is provided so that input sequences which do not correspond to a file can have a meaningful

identifier in the host. The first time the Read sequence is called, this field will be the null-string.

read_message Fill this field with any additional textual information you want to provide during the Read sequence. For example, you may want to specify the codec of a QuickTime movie in this field. The first time the Read sequence is called for a given input sequence, this field will be the null-string.

start_smpte_frames, time_base, reel_name

These fields are currently ignored by After Effects, but you may find them useful. (They will stick around for the current instance of the input sequence, but they are not saved out to disk. If you need such information to be more permanent, store it in your revertInfo handle in the FormatRecord structure and it will be saved to disk.)

read_time In the Read sequence, this structure will be set to the time of the desired frame.

read_dur In the Read sequence, this structure will be set to the duration of the desired frame. This would allow, for example, a plug-in which could integrate multiple frames temporally.

read_rect In the Read sequence, this structure will be set to the entire buffer.

write_name Fill this field with the name of your output sequence during the Options sequence. This is provided so that output sequences which do not correspond to a file can have a meaningful identifier in the host. The first time the Options sequence is called, this field will be the null-string.

write_message Fill this field with any additional textual information for output during the Options sequence. For example, you may want to specify the codec of a QuickTime movie in this field. The first time the Options sequence is called for a given input sequence, this field will be the null-string.

frame_num_to_add During the Write sequence this field will contain the frame number which is to be added to the output sequence. If it's set to -1, add this frame as the next frame in the file.

frames_to_add During the Write sequence this field will contain the duration of the frame to be added (in frame numbers at the current frame-rate.)

was_compressed Set to TRUE during the write sequence if the image data being passed has previously been compressed and decompressed. Some compression schemes want to know this.

origin_h, origin_v Reserved for future use.

alpha_label This structure will contain a description of the alpha channel information of the buffers handed to the Write sequence. You can use this for labeling purposes, or you may need to know alpha information to conform the output to your device or file format.

interlace_label This structure will contain a description of the field information of the buffers handed to the Write sequence. You can use this for labeling purposes, or you may need to know field order information to conform the output to your device or file format.

Calling Sequences

The FXIF specification uses the same prepare-start-continue-finish calling sequence that the 8BIF specification provides. FXIF plug-ins must support all of the calling sequences that 8BIF plug-ins support — read, estimate, options, and write.

The FXIF specification adds four more calling sequences. These new calling sequences follow the same conventions as the 8BIF calling sequences. See “Calling Sequences” in the Adobe 8BIF plug-in specification for details on the four stages of each calling sequence.

One sequence, the “Setup” sequence, is called at the launch of the host. This gives the plug-in the chance to initialize communication with hardware, set up globals, or anything else that the plug-in writer wants to do.

At the close of the host’s session, the “Setdown” sequence is called. This gives the plug-in the chance to clean up any operations that were performed in Setup.

The “Read” sequence remains essentially the same. One difference is the plug-in is provided with time information so it can read a specific frame from a file or device. Additional fields in the TimeInfo handle are provided so the plug-in can communicate time-based and other descriptive information back to the host.

The “Estimate” sequence is unchanged from the 8BIF specification.

The “Options” sequence is slightly different in After Effects. After Effects may need to display options information about an output sequence without actually bringing up the dialog. It does this by calling the Read-Start-Prepare sequence for Options without calling the Continue selector. This gives the plug-in the chance to translate its revertInfo handle (and other state information) into a form that After Effects can understand and display to the user. A true “options” sequence will call all the Continue selector, where you can bring up your dialog. As an example, consider a non-file based plug-in. The standard 8BIF specification is at a loss for the name of the output, because the name of the output is the name of the file. The FXIF mini-options sequence gives the plug-in a chance to fill in the blanks.

Two additional sequences have been added to accommodate writing multiple samples to files. Before a writing session begins, the “BeginAdd” sequence is called. This gives the plug-in a chance to do any initialization to the file or device before any video information is added.

The “Write” sequence remains the same except the plug-in is given the time and duration of the frame to add.

The “EndAdd” sequence allows the plug-in to finish an output sequence. The plug-in should clean up any operations performed in “BeginAdd.”

Other notes

A bug in After Effects 2.0.2 prevented error values returned from the Write sequence from halting the rendering process. This was fixed in After Effects 3.0.

The ProcessEvent callback is not supported in After Effects 3.0.

7 Special Considerations

The Adobe After Effects plug-in specification can be a bit intimidating, especially if you're writing a plug-in from scratch. This section describes a few things you should look out for when writing an effect — those parts of the specification which are especially subtle, tricky, or unique to the After Effects paradigm.

Extent Rects

If you use the extent-rectangle information to speed up processing of your effect (and you should, if you can) be sure to tell After Effects by setting PF_OutFlag_USE_OUTPUT_EXTENT. If you use in_data->extent_hint, set them both; this rectangle is the intersection of the output and input extent rects. If these flags are set improperly, caching of effect output will not be as efficient.

Before testing your extent-rect code, disable the effect cache. Hold down the option key and choose General Preferences from the Edit menu. A checkbox should appear which allows you to disable the layer cache. After Effects will then render your effect whenever anything in your composition changes. This ensures that After Effects re-renders your effect each time one of the extents changes; otherwise, there may be cases where your code doesn't work and AE's caching mechanism will obscure them.

Move the layer within the composition boundaries such that it is cropped by the composition. The output->extent_hint is portion of the layer which is visible in the composition. Add a mask to your layer and move it around. This changes the params[0]->u.ld.extent_hint, which encloses all of the non-zero alpha areas of the image.

The in_data->extent_hint is the intersection of these two rectangles. It (potentially) changes whenever either of the above rectangles changes.

If you write an effect that resizes its output buffer (smaller or larger), extent rect handling can be kind of tricky. Extent rectangles are computed in the coordinate space of the original input layer, before its resizing and origin shifting. If you keep that in mind, that can simplify things somewhat.

Another thing to consider when writing a "resizer" effect is that PF_World's are limited to 4000x4000 pixels. Make sure to account for the downsampling factor when you compute the output size — users should be able to render a full resolution version without the output buffer exceeding 4000x4000. If it does, you can either clamp it to that size or issue an alert dialog. (See "Error Handling" for details)

Once you're certain that your code behaves properly, re-enable the cache and see how frequently the effect needs to re-render. If you have a particularly slow effect, you may want to alter the extent flags that you set. (An example is the Drop Shadow effect. Because users frequently apply a static drop shadow to a still image, it was decided that the output extent-hint would be ignored so that the cache would be hit more often.)

Alpha Channels

After Effects 2.0 supported two alpha modes – straight mode and premul mode. This was selected by checking the “Extra Color Precision” checkbox in the General Preferences dialog box. In After Effects 3.x, “Extra Color Precision” is always on, so now only straight mode is supported.

Parameter Situations

If your effect uses random functions, or varies based on an implicit parameter such as time, be sure to set the PF_OutFlag_NON_PARAM_VARY at GLOBAL_SETUP or else After Effects will mistakenly cache your effect’s output with still footage items, such as PICT files.

If you use the value of any parameter at a different time, such as a video from another time, set the PF_OutFlag_WIDE_TIME_INPUT at GLOBAL_SETUP time. Otherwise, After Effects’ caching mechanism could bite you.

If your slider doesn’t seem to be usable, look at the valid_min, slider_min, valid_max and slider_max fields. Is it a fixed slider? If so, did you convert your min’s and max’s to reasonable fixed values? (I don’t know how many times I’ve set def.u.fd.valid_max = 100, which of course means that the largest possible value is about 0.0015.)

If you use sliders (or other controls) that have explicit or implicit pixel dimensions in them, account for the downsample factor in your calculations. Test at 1/2, 1/4, and custom resolutions and compare the output. Try differing horizontal and vertical downsample factors too. The “resizer” sample effect shows you what happens when you properly and improperly deal with different resolutions.

Be careful if one of your parameters is a speed or velocity parameter. Consider the ripple effect. It assumes a constant and uses the current time to determine how far along the ripple has gone. (d = v * t.) Now if the user interpolates the speed over time, the proper thing to do would be to integrate the velocity function from time zero to the current time. Ripple does not do this, but provides a “phase” parameter that the user can interpolate to her heart’s content instead, providing correct results as long as the speed is set to zero. If you want to provide the correct behavior, you can sample (and integrate) the speed parameter from the beginning of time until the current time using PF_CHECKOUT_PARAM, or you can provide a “phase” or “distance” parameter and warn the user about interpolating the speed. We've found that the speed of checking out many parameter values is pretty negligible compared to rendering and therefore recommended for cases like this.

If you check out parameter values at other times, or use layer parameters at all, please be sure to check in a parameter when you’re done with them, even if an error has occurred since you checked it out. And remember, checked out parameters should be considered read only.

Sequence Data

If you use sequence data you might want your handles to be “flat” because After Effects will write sequence data out to disk with a project. If you

choose to have pointers or other handles inside your sequence data, set the PF_OutFlag_SEQUENCE_DATA_NEEDS_FLATTENING flags at GLOBAL_SETUP time. See the comment in AE_Effects.h under “Output Flags” for a full description of the flatten/unflatten situation.

If your global data changes from invocation to invocation, be sure to set the PF_OutFlag_GLOBALS_ARE_CHANGABLE flag at GLOBAL_SETUP time.

Error Handling

Be sure to check error codes from all of the PF callback routines. If an error is returned, it’s probably a user interrupt and you should abort the rest of the routine, but be sure to dispose of any PF_Worlds or other memory you’ve allocated. Also, don’t forget to check in any parameters you’ve checked out.

If an error is returned from PF_NEW_WORLD or a Macintosh Toolbox routine, you can report the error to the user by copying a string into out_data->return_msg and setting PF_OutFlag_DISPLAY_ERROR_MESSAGE.

Debugging

While developing your plug-in you’ll probably want to test it, make a few changes, and run it again. It is not necessary to quit After Effects each time. You can replace your plug-in code in the After Effects plug-in folder and ask After Effects to reload it by pressing the Control and Clear keys simultaneously. This won’t load any new plug-ins, it just asks After Effects to reload any plug-ins that were present when After Effects started up.

Be Responsive

Try and make your plug-ins as responsive as possible, call Abort and Progress frequently. This is reasonably safe because After Effects will check and return immediately if it thinks you’re calling them to often, but it’s still a good idea not to over do it!

Adding Parameters

A fairly common problem is failing to completely clear out a PF_ParamDef structure prior to filling it in and calling PF_ADD_PARAM. If you (or your users) see any of the following After Effects messages when applying your filter:

> Effect control conversion problem
> Problem with Effect Control
> Some of the Controls of the effects will be reset

or you find your filter loosing some of its settings, then this is likely the problem. A simple solution is to call the macro AEFX_CLR_STRUCT with your PF_ParamDef structure as the only parameter, prior to filling it in. This

macro will clear the structure for you. This is demonstrated in the latest versions of the plug-in samples packaged with the SDK.

*gdeviceH . . . 45
alpha_label . . . 52
appl_id . . . 18
area . . . 45
cbs . . . 42
cgrafptr . . . 42
char_code . . . 45
COMP_TO_LAYER . . . 46
contextH . . . 41
continue_refcon . . . 44
current_frame . . . 45
current_time . . . 19
data . . . 27
depth . . . 44
downsample_x, downsample_y . . . 20
duration . . . 50
e_type . . . 41
effect_ref . . . 18
effect_win . . . 42
error_msg . . . 50
evt_in_flags . . . 42
evt_out_flags . . . 42
extent_hint . . . 20
extent_hint . . . 27
field . . . 19
fixed_fps . . . 50
flat_sdata_size . . . 23
frame_data . . . 23
frame_num_to_add . . . 51
FRAME_TO_SOURCE . . . 47
frames_to_add . . . 51
get_callback_addr . . . 37
GET_COMP2LAYER_XFORM . . . 46
GET_LAYER2COMP_XFORM . . . 46
GET_PLATFORM_REFS . . . 37
global_data . . . 22
global_data, sequence_data, frame_data . . . 20
horiz_offset . . . 46
in_flags . . . 20
index . . . 45
INFO_DRAW_COLOR . . . 47
INFO_DRAW_TEXT . . . 47
INFO_GET_PORT . . . 47
inter . . . 18
interlace_label . . . 52
key_code . . . 45
last_time . . . 44
LAYER_TO_COMP . . . 46

local_time_step . . . 19
magic . . . 42
modifiers . . . 44
modifiers . . . 45
my_version . . . 22
name . . . 22
num_clicks . . . 44
num_params . . . 18
num_params . . . 22
origin_h, origin_v . . . 51
out_flags . . . 23
output_origin_x, output_origin_y . . . 20
param_title_frame . . . 46
PF_ABORT . . . 28
PF_ACOS . . . 37
PF_ADD_PARAM . . . 28
PF_AREA_SAMPLE . . . 32
PF_ASIN . . . 37
PF_ATAN . . . 37
PF_ATAN2 . . . 37
PF_BLEND . . . 32
PF_CEIL . . . 37
PF_CHECKIN_LAYER_AUDIO . . . 30
PF_CHECKIN_PARAM . . . 29
PF_CHECKOUT_LAYER_AUDIO . . . 30
PF_CHECKOUT_PARAM . . . 29
PF_Cmd_ABOUT . . . 13
PF_Cmd_DO_DIALOG . . . 14
PF_Cmd_EVENT . . . 15
PF_Cmd_FRAME_SETDOWN . . . 15
PF_Cmd_FRAME_SETUP . . . 14
PF_Cmd_GLOBAL_SETDOWN . . . 13
PF_Cmd_GLOBAL_SETUP . . . 13
PF_Cmd_PARAMS_SETUP . . . 14
PF_Cmd_PARAMS_UPDATE . . . 15
PF_Cmd_RENDER . . . 15
PF_Cmd_SEQUENCE_FLATTEN . . . 14
PF_Cmd_SEQUENCE_RESETUP . . . 14
PF_Cmd_SEQUENCE_SETDOWN . . . 14
PF_Cmd_SEQUENCE_SETUP . . . 14
PF_CONVOLVE . . . 33
PF_COPY . . . 33
PF_COS . . . 37
PF_DISPOSE_WORLD . . . 36
PF_Event_ACTIVATE . . . 43
PF_Event_CLOSE_CONTEXT . . . 43
PF_Event_DEACTIVATE . . . 43
PF_Event_DO_CLICK . . . 43
PF_Event_DRAG . . . 43

PF_Event_DRAW . . . 43
PF_Event_IDLE . . . 43
PF_Event_KEYDOWN . . . 43
PF_Event_NEW_CONTEXT . . . 43
PF_EXP . . . 38
PF_FABS . . . 38
PF_FILL . . . 33
PF_FLOOR . . . 38
PF_FMOD . . . 38
PF_GAUSSIAN_KERNEL . . . 33
PF_GET_AUDIO_DATA . . . 30
PF_HLS_TO_RGB . . . 39
PF_HUE . . . 39
PF_HYPOT . . . 38
PF_ITERATE . . . 34
PF_ITERATE_LUT . . . 35
PF_ITERATE_ORIGIN . . . 35
PF_LIGHTNESS . . . 40
PF_LOG . . . 38
PF_LOG10 . . . 38
PF_LUMINANCE . . . 39
PF_NEW_WORLD . . . 36
PF_OutFlag_CUSTOM_NTRP . . . 26
PF_OutFlag_CUSTOM_UI . . . 26
PF_OutFlag_DISPLAY_ERROR_MESSAGE . . . 25
PF_OutFlag_I_AM_OBSOLETE . . . 27
PF_OutFlag_I_DO_DIALOG . . . 25
PF_OutFlag_I_EXPAND_BUFFER . . . 25
PF_OutFlag_I_SHRINK_BUFFER . . . 26
PF_OutFlag_I_USE_AUDIO . . . 26
PF_OutFlag_I_USE_SHUTTER_ANGLE . . . 26
PF_OutFlag_I_WRITE_INPUT_BUFFER . . . 26
PF_OutFlag_KEEP_RESOURCE_OPEN . . . 24
PF_OutFlag_NON_PARAM_VARY . . . 24
PF_OutFlag_NONE . . . 24
PF_OutFlag_NOP_RENDER . . . 26
PF_OutFlag_PIX_INDEPENDENT . . . 26
PF_OutFlag_REFRESH_UI . . . 26
PF_OutFlag_SEND_DO_DIALOG . . . 25
PF_OutFlag_SEND_PARAMS_UPDATE . . . 24
PF_OutFlag_SEQUENCE_DATA_NEEDS_FLATTENING . . . 24
PF_OutFlag_SQUARE_PIX_ONLY . . . 26
PF_OutFlag_USE_OUTPUT_EXTENT . . . 25
PF_OutFlag_WIDE_TIME_INPUT . . . 24
PF_OutFlag_WORKS_IN_PLACE . . . 26
PF_Param_ANGLE . . . 21
PF_Param_CHECKBOX . . . 21
PF_Param_COLOR . . . 21
PF_Param_LAYER . . . 20

PF_Param_POINT 21
PF_Param_POPUP 21
PF_Param_SLIDER, PF_Param_FIX_SLIDER 21
PF_POW 38
PF_PREMUL 36
PF_PREMUL_COLOR 36
PF_PROGRESS 28
PF_REGISTER_UI 29
PF_RGB_TO_YIQ 39
PF_SATURATION 40
PF_SIN 38
PF_SPRINTF 38
PF_SQRT 38
PF_STRCPY 38
PF_SUBPIXEL_SAMPLE 32
PF_TAN 38
PF_TRANSFER_RECT 35
PF_TRANSFORM_WORLD 36
PF_YIQ_TO_RGB 39
pixel_aspect_ratio 20
platform_ref 28
plugin_state 42
poster_time 50
qd_globals 19
quality 18
read_dur 51
read_message 51
read_name 50
read_rect 51
read_time 51
reserved_flt 42
return_msg 23
RGB_TO_HLS 39
rowbytes 27
screen_point 44
screen_point 45
send_drag 44
sequence_data 23
serial_num 18
shutter_angle 19
SOURCE_TO_FRAME 47
start_smpte_frames, time_base, reel_name 51
time_info 49
time_info_version 50
time_scale 19
time_selector 49
time_step 19
total_time 19
u 42

update_rect 44
utils 18
version 18
w_type 42
was_compressed 51
what_cpu 18
what_fpu 19
when 44
when 45
width, height 19
width, height 27
width, height, origin 23
world_flags 27
write_message 51
write_name 51

PostScript error (--nostringval--, --nostringval--)