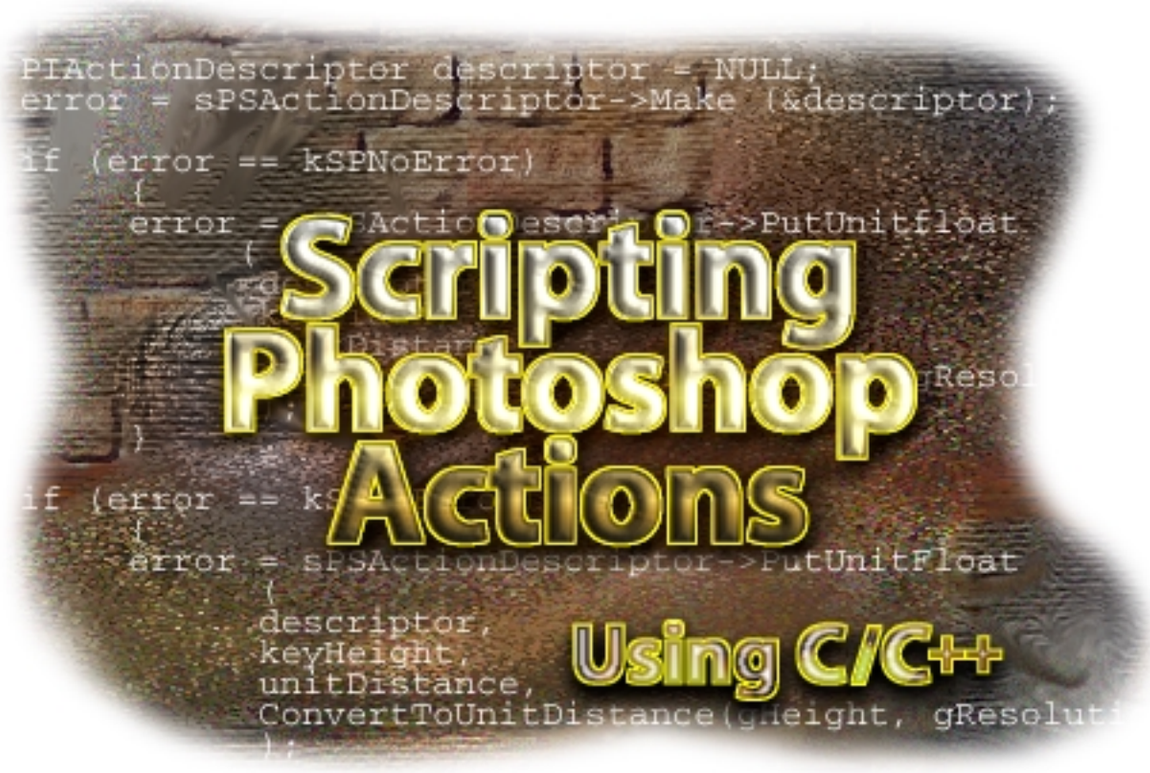


Adobe® Photoshop® 5.5 Actions Event Guide

Version 5.5 Release 1
August 1999

Adobe Action Events Guide

The Photoshop Object Model was conceived and implemented by Sean Parent. This document was written by Chris Bailey based on interviews and written materials from Sean Parent and extensive interviews and reviews by Andrew Coven. In addition, this document was edited and reviewed by Chris Bailey, Bobby Smith, Andrew Coven, Michael Snyder, John Long, and Sean Parent.

Version History

Date	Author	Status
30 July 1997 thru 29 August 1997	Foster T. Brereton	First Draft. All text in Red is unimplemented, needs major revision or is, in any other way, utterly unfinished.
28 August 1997	Foster T. Brereton	Moved the project to FrameMaker 5.5 Format, broke the link in the Appendixes from referencing external files.
25 January 1998	Chris Bailey	Second Draft. Updated Format to current Adobe Standards. Created new content covering the Photoshop Object Model, Automation syntax, and Appendixes.
12 March 1998	Chris Bailey	Third Draft. Extensive update to include step-by-step development of automation plug-in.
6 April 1998	Chris Bailey	Corrections and final updates to tabular materials.
5 August 1999	Tina Wu	Added two chapters for using the listener plug in.

Title Page 1

Version History 2

Table of Contents 3

1. Introduction 6

About This Guide 6

Audience 7

Getting Started 7

General Definitions 8

The Actions Palette 8

Actions 8

The Photoshop Object Model 8

Programmable Elements 9

Underlying Naming Conventions 9

C Language Photoshop Event Definitions 9

Event Names: Keys /ID Codes and Name Macros 11

The Descriptor Block 11

Summary 12

2. Object Model 13

Photoshop Containment Structure 13

Photoshop Inheritance Model 14

Element Hierarchy 15

Color Inheritance Hierarchy 16

Mode Inheritance Hierarchy 16

Adjustment Inheritance Hierarchy 17

Format Inheritance Heirarchy 17

Actions and Targets 18

Targets 19

Handling User Interaction While Playing an Action 19

Elements, Classes, References, and Target Paths 21

3. Creating An Automation Plug-in 22

Standard Plug-ins vs. Automation Plug-ins 22

The Photoshop/Automation Plug-in Relationship 23

Photoshop 5.0 Organization 24

Plug-In Component Architecture (PICA) 24

Plug-in Data Structures (PiPLs and Suites) 25

PiPLs 25

Suites 26

Handling Suites In Your Plug-in 27

Photoshop Suite Definitions 29

The Plug-in Development Process 31

Our new automation plug-in 33

Examining the TriggerFilters plug-in project 34
Writing Your Own Plug-in: Step-by-Step 38
A Brief Note on ADM and Dialog Element Management 57
1) Edit the PiPL 59
2) Editing dictionary resources 61
3) Edit the Dialog resources 63
4) Edit the String Resources 65
Other Good Information: 76

4. Using Listener 77

Listener Does Most Of The Work 77
Listener Creates The Code 77
How To Develop An Automation Plug-in 78
Step 2: Clear the Listener.log file 78
Step 3: Prepare the Environment 78
Step 4: Apply Desired Actions On Test Document 78
Step 5: Build Off An Existing Automation Plug-in: MakeNew 79
Step 6: Verify The Contents Of Listener.log File 79
Step 7: Incorporate Function Calls 80
Step 8: Paste Function Definitions and Function Prototypes 80
Step 9: Comment Out The MakeNew Actions 81
Step 10: Build Your Plug-in And You Are Done! 81

5. More Listener 82

Type Tool 82

6. Filter Action Events 84

Built-In Filter Events 85
Plug-in Filter Events 96

7. Tool Classes 114

Built-In Tool Classes 115
Marquee Tool Selection Classes 115
More Tools 117

8. Element Action Events 120

Built-in Element Events 121

9. Document Events 128

Built-In Document Events 129

10. File Action Events 132

Built-in File Events 133

11. Classes and Formats 134

Classes 135
Formats 160

12. Types and Enumerations 166

Types and Enumerations 166

13. Automation Plug-ins 176

A. Information Sources 179

Common prefixes and conventions 180

B. PIUBasic Files 181

C. Development Process 182

D. Key Constants 185

Key Constants 185

E. Event Constants 198

Event Constants 198

F. Enumerated Constants 202

Enumerated Constants 202

G. Pin Ranges 220

About Pin Ranges 220

H. Glossary 226

Photoshop Object Model Definitions 226

Photoshop Plug-in Terminology 227

Index 229

1. Introduction

Welcome to the Adobe Photoshop® Actions Event Guide!

This document contains the information needed to write Photoshop 5.0 automation plug-ins using C. It defines the underlying Photoshop Object Model and related terms and lists each scriptable function and associated parameters with examples. In addition, it briefly explains the basic C syntax used in writing Photoshop automation plug-ins and illustrates some examples that the reader may use as templates to create custom automation plug-ins.

About This Guide
This chapter provides a brief overview of the content and the basic terminology for this guide.

Chapter Two documents the Photoshop Object Model and how the automation system manipulates events and objects.

Chapter Three takes a step-by-step approach to explaining how to create a Photoshop automation plug-in. It contains explanatory material on what's inside an automation plug-in and introduces the Photoshop specific automation functions, features and syntax.

Chapters Four through Ten provide the specific details on how to manipulate each type or class of Photoshop elements.

Chapter Four covers Filters Events (Built-in Filters – Blur, Emboss, Feather, Invert, Offset, Unsharp Mask, etc.; and Plug-in Filters – Angled Strokes, Bas Relief, Crosshatch, Grain, Palette Knife, Pointilize, Texturizer, Wave, etc.).

Chapter Five covers Tools such as Marquee/Selection, Pencil, Airbrush, Smudge, Path, etc.

Chapter Six covers Element Action Events including Add, Copy, Cut, Define Pattern, Duplicate, Group, Make, Move, Select, Transform and Undo.

Chapter Seven covers Document Events such as Canvas Size, Convert Mode, Crop, Export, Flatten Image, Print, Revert and Save.

Chapter Eight covers the File Action Events Import, Open, and Rasterize.

Chapter Nine covers Classes and Formats. Classes include Color, RGB Color, CMYK Color, Element, Document, Pixel, Layer, Hue/Saturation,

Channel, Format, Inport and Export. Formats include JPEG, Photoshop EPS, PICT file, Raw, TIFF, GIF89A Export, BMP, and others.

Chapter Ten covers Types and Enumerations.

The Appendixes document the additional sources of information and list in reference form specific Enumerated Values, Key Constants, Event Constants, and Variable Pin Ranges.

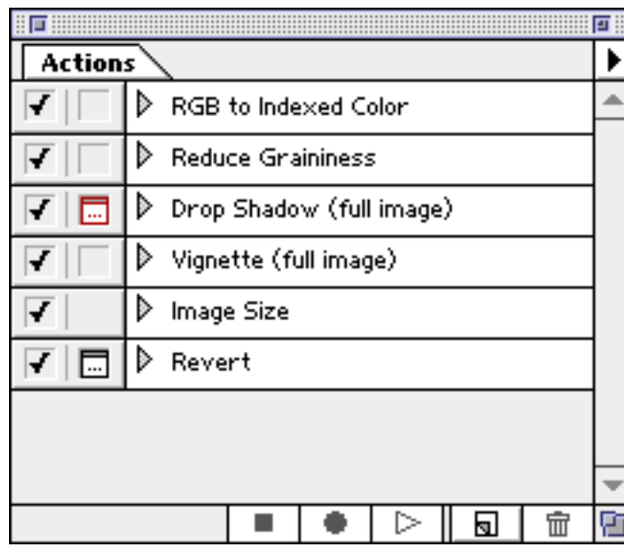
Audience

This document is intended for C programmers who want to fully automate Photoshop functions using C. With Photoshop 5.0 and its fully programmable Actions Engine, you can now achieve full programmable automation.

You are expected to have a working knowledge of Adobe Photoshop 5.0 and to understand how internal and plug–in features and filters work from a user's point of view. You should first read the *Photoshop 5.0 API Guide*. This guide assumes you understand Photoshop terminology such as *paths*, *layers* and *masks*. For more information, consult the *Adobe Photoshop User Guide*.

Getting Started

Starting with Photoshop 4.0, Photoshop permitted users to manually automate Photoshop actions. The action palette provides a visual mechanism for users to record, store and replay sequences of Photoshop actions.



However, user interaction is required at each step of recording and playing. Starting in Photoshop 5.0, automation is further enabled by allowing a new type of plug-in: the automation plug-in. Unlike previous plug-ins that were restricted to a simple interaction between the host application (Photoshop) and the plug-in, an automation plug-in can interact with other plug-ins and any actionable Photoshop event. This permits the creation of powerful, complex functions that fully utilize internal Photoshop resources.

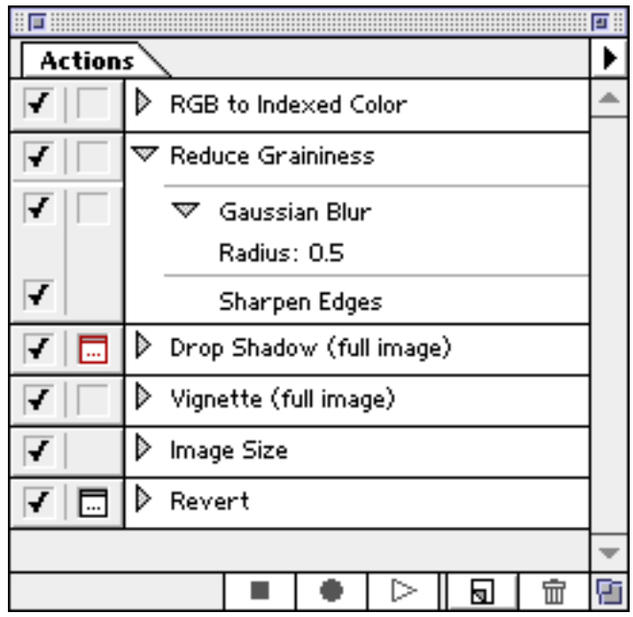
The mechanism that permits this interaction is the Actions Engine and its event name/descriptor block structure. To successfully manipulate Photoshop elements and events, it is important for you to understand the underlying Photoshop Actions Object Model and how the various elements, objects, classes, and action events behave and interact.

General Definitions

The Actions Palette

In Photoshop 4.0 and later, the user can manually record a sequence of keyboard and/or mouse commands and call this sequence an action. Actions can be played, edited and deleted. Each sequence is named and appears in the Action Palette as a individually selectable “action.”

The Actions Palette holds the list of available actions and a VCR-like control panel with “play/stop,” “record,” “pause,” and “delete” buttons. Within each action, individual commands in the sequence can be shown (see below).



Actions

Internally, Actions consist of Photoshop events and targets of those events. An action can be a single event or a sequence of events. Events themselves are individual Photoshop commands or instructions that act on elements or objects. Every Photoshop event has a data structure associated with it that the automation plug-in uses to manipulate the event and target selection.

The Photoshop Object Model

Underlying the Photoshop user interface and enabling the Actions mechanism is the Photoshop Actions Object Model. Like other object oriented programs, Photoshop is organized in hierarchical *elements* that include *objects*, *classes*, and *functions*. Actions are constructed from filters and commands (events and functions) with parameters that act on target selections (elements). Targets are specified by references or implied

references through an object container hierarchy and parameters are organized in classes.

None of this organization is particularly visible to the end user of the program. However, once the underlying architecture of Photoshop is understood, there are clear connections between the hierarchical structure of Photoshop, the scripting syntax and the standard user interface.

The object model structure allows the creation of plug-ins that can access Photoshop features and manipulate user image files. The model defines programmable and non-programmable elements, program features and functions, data classes, image formats and external operators. It is a rigid, but flexible structure that enables easy programming of complex events by sequencing several discrete events and passing a wide range of parameters to individual functions within a sequence.

Programmable Elements

Programmable elements are objects and classes that represent a tangible object inside Photoshop. Elements can be grouped into filters, tool classes and events, element action events, document events, file action events, and classes and formats. Each element is unique with its own data types and functional capabilities, and elements can contain other elements.

Underlying Naming Conventions

Names, Descriptors, and Key/Value Pairs

Wherever possible, the same naming conventions used in the *Plug-in Resource Guide* will be used. Elements used to describe individual functions or parameters include names, parameters, descriptors, and key/value pairs.

Every programmable Photoshop event has the following elements that define it: an event name, a target (implied or explicit), and parameters (optional). These elements are accessed internally (and for external plug-in development) using the C language terminology explained in this document and the *Photoshop 5.0 API Guide*.

C Language Photoshop Event Definitions

In Photoshop 4.0, every common Photoshop programmable event (command, feature or filter) was assigned a unique four character string as an event name. This four character string was used as a hash key to look up the corresponding internal Photoshop event. When creating third-party filters and plug-ins, developers are required to create a unique event name. Starting with Photoshop 5.0, new Photoshop event names, such as those for new internal Photoshop commands and new third-party filters, may be strings of any length as long as they are unique. For this document, event names are assumed to be four character strings that uniquely define each command, filter or function.

NOTE: To simplify C programming, event name macros that more closely resemble their function have been defined. For example, the macro: `eventGaussianBlur` may be used to access the event Gaussian Blur. These macros evaluate to the four character event names. In the documentation, you may see references to event names such as `eventGaussianBlur`, but note that in every case, when programming in C, these names are actually evaluated to their four character names. In the case of Gaussian Blur, the event key ID is `'GsnB'`.

Every event can have a corresponding target object. The target can be explicitly defined or, if not defined, it is assumed to be the last selected object. For example, a Gaussian Blur event without a specified target will be performed on the current selection of the current layer of the current document.

Events can also have optional parameters that are used to specify the parameters of the current event. For example, the specific parameters for a given Gaussian Blur define the characteristics of that particular blur event, such as `KeyRadius = 5`.

Internally, Photoshop maintains a database of event, objects, classes, and parameter types, and uses unique 4 character IDs to select a specific command, feature, filter or function.

In addition to parameter ID keys, Photoshop identifies the type of data to be used in the event. For example, before passing a floating point value in an event call, the Floating Point Data Type ID key, `typeFloat`, would be used to flag the data as a floating point number.

Below is the general format for accessing programmable events:

Event Name/Parameter Conventions

Predefined ID name (macro Event Name)	Actual 4-character code ('event ID')
`eventName`	`'Name'`

General Format

```
eventName ("Name")
      {
      keyTarget ("trgt")   reference     //optional TARGET reference
      keyName ("key ID")   [pin values]  //optional key/value pair
      ('key constant')     [pin values]  //additional key/value pairs

      }
```

Descriptor Block

Example:

Event: A Gaussian Blur of radius 2.6

Event ID/Name: `eventGaussianBlur ("GsnB")`
Descriptor Block:
Parameter:

```
    {
    keyRadius ("Rds ")  'typefloat ("Flt ")  2.6
    }
```

Target: In this case, there is no explicit target, so the implied target is the current selection, on the current layer, in the current document.

Parameters: There is one parameter item in the Descriptor Block. It provides the parameter for the radius amount (required for the Gaussian Blur filter). In this case, there is one key ID and one type ID. The key ID is 'Rds ', the ID for the key "Radius Amount," and the other is 'Flt ', the ID for the type "floating point." The final parameter is the radius value: 2.6 pixels.

The event name consists of a unique four-character constant (future events and plug-in filters can have longer name constants) with an optional target and parameter list. The first item in the list is the explicit target (if any) with additional parameters following.

Event Names: Keys /ID Codes and Name Macros

As noted above, all elements and “actionable” behaviors of Photoshop, including commands, filters, objects and classes, have a unique four letter constant or “key” associated with them. While the ID code or key indicates the command or function, the specific parameters passed with the event define the specific Photoshop selection, element or function being acted upon as well as the actual parameters of the function.

The Descriptor Block

The Descriptor block is a list of variables in the form of <key><value>. The first item in the Descriptor Block is the target of an event (if any). If no explicit target is given, the target is implied to be the current document or the first element in the target container chain that understands and can respond to the event (see Chapter Two).

Descriptors allow Photoshop to pass the parameters associated with each event back and forth from the host to the calling plug-in. Since the Descriptor Block can contain as many key/value pairs as necessary, it provides the structure to allow Photoshop to perform an action of any length and containing any number of commands and parameters.

The Descriptor Block provides a consistent and yet flexible mechanism to allow the many actionable filters and plug-ins to control Photoshop precisely. Note that a key/value pair can hold many different items including integers, objects, floating point values and references to other objects. Generally the key/value pair will refer to a specific Photoshop feature and the parameters associated with that feature.

The value parameters in a Descriptor Block can be any value in the range that is defined by Photoshop for the given function or element. For example, for a call to the Gaussian Blur Filter, the legal values for the radius value used to calculate the amount of blur can be within the values of 0.1 and 250 (the same values available to a user accessing this filter manually). For built-in functions, the legal range of these values are set by the Photoshop program itself. For plug-in functions, the legal values appropriate to the function may be provided by the plug-in. Photoshop has functions that can correctly “pin back” illegal values to the maximum or minimum values (thus the name, pin value), but it is good practice to check values within your plug-in code.

Values can be integers, floating point numbers, unit floating point numbers, enumerated values, Boolean values, references to other objects, and aliases to files.

Summary

Photoshop 5.0 and its underlying Event Name/Descriptor Block structure is robust and offers a great deal of flexibility in automating image processing.

Underneath its familiar user interface, Photoshop 5.0 is organized as a large, accessible database of commands, filters, and elements, each with unique keys and a variable list of parameters. You can write a plug-in that automates Photoshop and accesses commands, filters, elements and other plug-ins, passing their specific parameters to them.

This chapter briefly introduces the internal structure of Photoshop. The next chapter explains the Photoshop Action Object Model in detail.

2. Object Model

The Photoshop object modeldetermines what actions operate on which objects and what parameters are associated with each element. Two key Photoshop concepts are *containment structures* and *inheritance hierarchies*. These structures reveal how Photoshop organizes its information about documents, layers, channels, selections, commands, filters, and action events.

Containers define what elements hold other elements, while inheritance hierarchies define what parameters or properties are available to the objects or subclasses held in containers. Containment structures hold independent objects that may or may not inherit properties from the container. On the other hand, inheritance hierarchies define what properties are inherited from one subclass to another.

For example, the Photoshop application (a container) can hold a document with many layers, each with their own properties. None of the layers necessarily share the properties of the document (guides or channels) or the application (menus, etc.).

On the other hand, elements in inheritance hierarchies do inherit the properties or parameters from the elements or classes above them in the inheritance chain. For example, the Mode inheritance class (RGB color, Grayscale, Bitmap, etc.) defines the possible channels within a document and what they represent. Inherited values define the subclass parameters and characteristics.

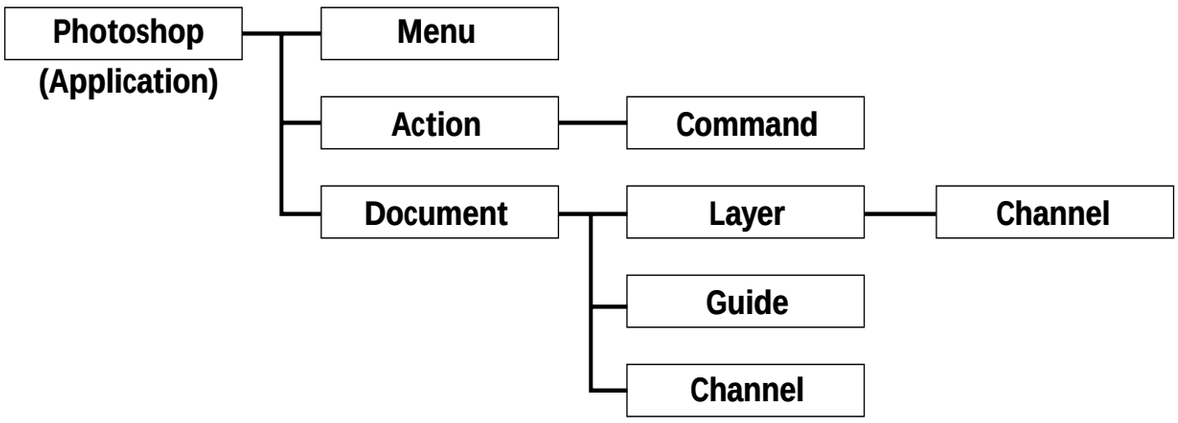
Note: All of the base classes in the inheritance hierarchies are in fact abstract classes, so deriving objects from them is impossible. And keep in mind that some of the base classes will require general parameters to exist in all classes derived from them.

Photoshop Containment Structure

As shown in the diagram below, the highest level container is the Photoshop application itself. At the first level, Photoshop can hold menus, actions, and a document (the user file). A document can hold layers, channels, guides, etc. A special case is the channel object which can be held in a document by itself, or as part of a layer. Any given selection will be within a channel or layer within a document. An action, event, command or function will “act” on a selection in the document. Selections can be explicitly defined or assumed to be the current or last defined selection.

This figure shows the Photoshop object containment structure.

Containment Hierarchy



The containment hierarchy shows all the elements that can contain other elements. Note that some elements are required to contain some, but not all, of the other elements. For instance, all documents are required to have at least one layer, but a document does not necessarily require the presence of a guide. Along the same lines, an element can contain more elements than required. Layers must contain at least one channel, for example, but can hold many more than the minimum.

Photoshop Inheritance Model

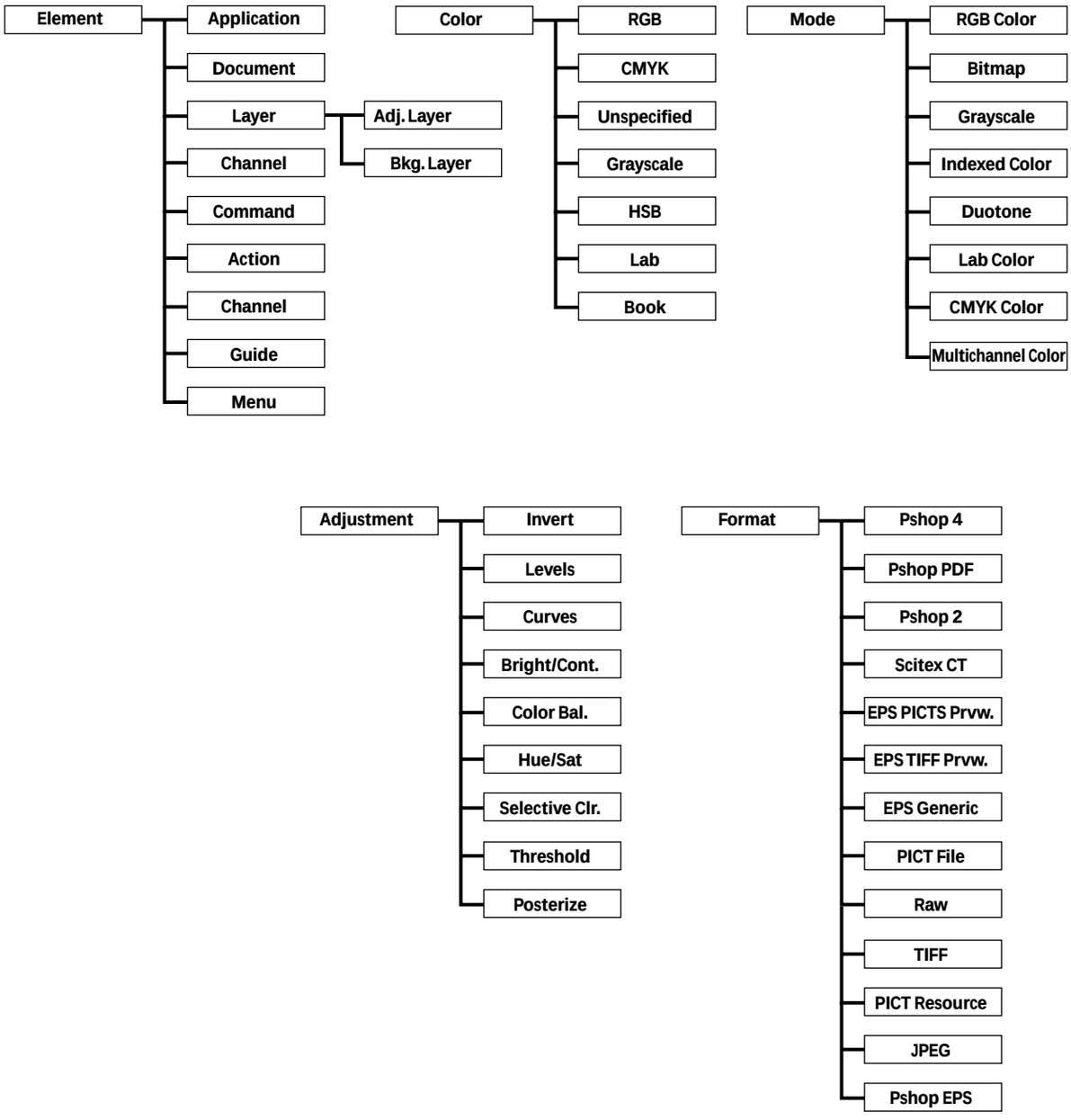
The Photoshop inheritance hierarchy gives structure to the characteristics expected of an object, element, class or selection.

For example, an object with the class “Color” will inherit the properties from the subclass under color (RGB, CMYK, Unspecified, Grayscale, HSB, Lab, or Book) and so forth.

Whenever an action is performed on an object, Photoshop works its way up the target chain, starting with the currently selected object, until it finds a container with the appropriate object.

The following diagram below shows the Photoshop Inheritance Hierarchies.

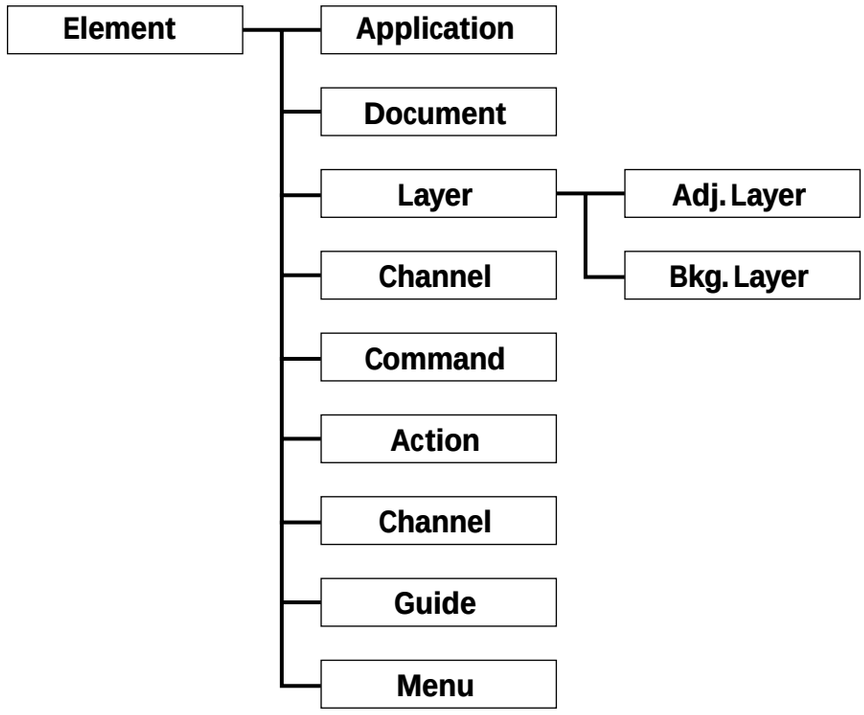
Photoshop Inheritance Structure



Element Hierarchy

The element inheritance figure shows the inheritance connection for elements. Following the rule that every Photoshop element inherits the parameters of its parent element or group, an adjustment layer object inherits the characteristics of its parent layer.

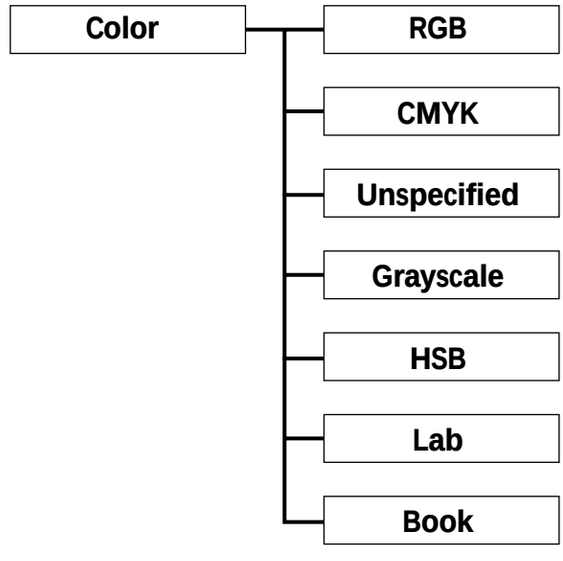
Element Inheritance



Color Inheritance Hierarchy

The color inheritance chart shows all the classes that are derived from the color base class. These objects typically contain integer values for each one of the channels present in that color scheme.

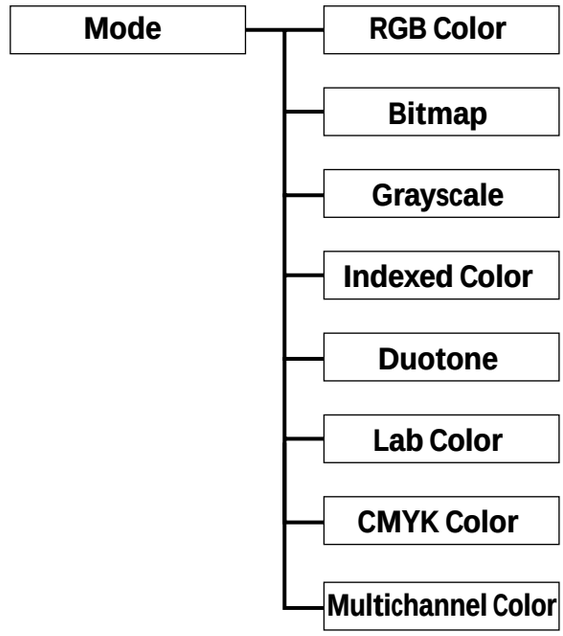
Color Inheritance



Mode Inheritance Hierarchy

Mode inheritance adjusts the channels and what they represent within the document. A document can only be one Mode subclass at any time.

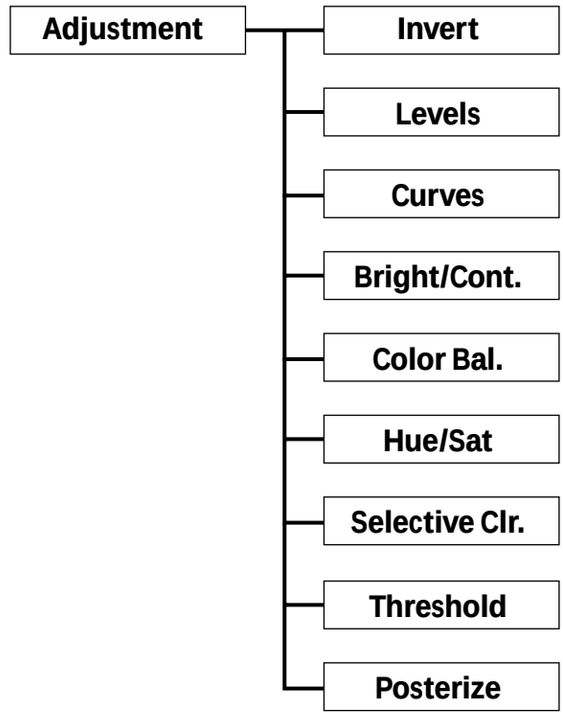
Mode Inheritance



Adjustment Inheritance Hierarchy

The adjustment inheritance diagram shows all the classes that are derived from the adjustment base class. Objects derived from the base are all the adjustment layer effects.

Adjustment Inheritance

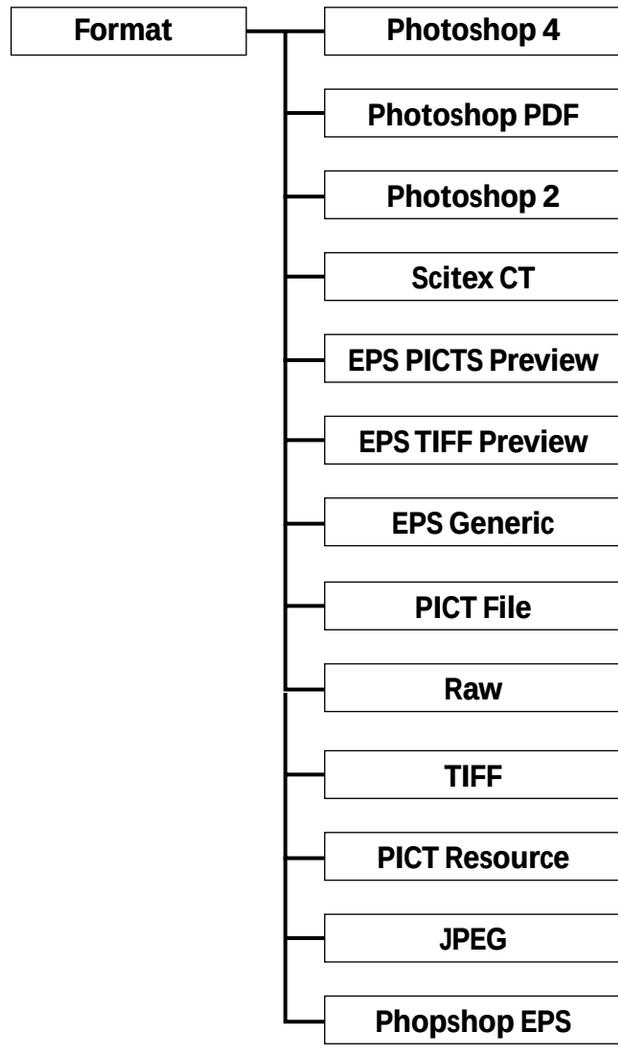


Format Inheritance Heirarchy

The format hierarchy defines all the possible types of files Photoshop can read and write. A file may be saved or loaded in any of these formats, and

this process can easily be automated to provide an action that can save multiple file formats with one command. There are also plug-in file formats.

Format Inheritance



Actions and Targets

The concepts of containment and inheritance come into play when considering an action and its target. The idea of containment is: a channel is contained in a document or a layer. A layer is contained in a document. A selection is contained in a layer, or a channel, etc. An action can “act” on a target, and that target can be specified in detail, i.e. document, layer, channel, selection, etc. But it is convenient to make some assumptions about the target where possible. For example, when executing a blur on a selection, it is convenient to assume that the blur will be on the current selection and leave it at that.

This is where the concept of inheritance comes in. When the Photoshop actions engine receives a command with no specified selection, it parses the request by moving up the containment chain until it finds an element that fits the target. For example, if a command is “Gaussian Blur with radius 2.6 pixels”, Photoshop moves up the chain of containers until it finds an element that understands the filter “Gaussian Blur” and then works its way down the chain of containers until it finds a layer or channel with a selection that can be “blurred.”

Targets

Targets are defined to be the frontmost item in Photoshop's current state, or the currently selected item in Photoshop's current state. In normal Photoshop usage, every time the user clicks on a different layer or moves to another document, the target changes.

Actions are performed on the current target, and it is possible to define other targets and attach them to descriptors for proper playback. You can switch targets, play an action, then switch again to perform another action on another target all by changing the target reference.

The reference to the current target, or the target that is going to be receiving the descriptor and event is defined in the first <key><value> item in the descriptor block. This special entry, `keyTarget`, is the first one Photoshop looks for in the descriptor block. If it finds a target there, then it will send the command to that target.

Targets are capable of receiving events and descriptors. But what happens if a target receives an action it is unable to perform? What if a layer gets an action to open a document? Clearly this is not something a layer is capable of doing, so it will pass the action to its container object. This means that Photoshop will search all of the objects starting with the current target until a container is found that can hold the object being manipulated. This link of container objects is called the *target chain*, and is always present in Photoshop. For instance, if you have a channel selected as the current target, then the target chain is Channel->Layer->Document->Photoshop (see the containment figure).

In the case of the layer receiving the command to open a document, the command will be passed up the target chain until one of those elements has the capability to handle it. In this case it is Photoshop itself that knows how to handle the open command.

The value in the target reference can be one of three things: a complete path, a partial path, or NULL. Each value type is handled differently.

If the reference is a complete path, then the target that is defined by that path receives the action, and all is well.

In the case where the reference is a partial path, the first target up the calling chain that can handle the path takes the command. This is important, because if you are trying to target channel "foo" of layer "snafu", it won't happen if a channel "foo" exists in layer "bar", and "bar" appears sooner in the target chain.

In the case where target value is NULL or does not exist, the first target up the target chain that can handle the command performs the action.

Handling User Interaction While Playing an Action

Handling user interaction during exceptions or unpredicted circumstances while playing an action is done by chosing one of three

modes under which an action can execute: UI Mode, No UI Mode, and Silent Mode. Each mode performs the given action, but provides information to the user at different times and in different ways.

UI Mode
UI Mode will execute an action and display the related dialog box for every step in the action. For instance, if you are including a filter in the current event it will bring up a dialog for that filter to confirm the values passed in by the descriptor block. This method is the most error-free, but will not allow you to automate without having to monitor your automation process, repetitively confirming dialog box values.

No UI Mode
No UI Mode is one step beyond UI mode, relying on the values passed in by the descriptor block and not requiring interaction. It may, however, display a dialog box if the routine comes across values that are invalid for execution and that cannot be coerced into valid values. It will also display dialogs for situations where a descriptor block component is incomplete. This mode provides higher automation capabilities with minimal user overhead. This mode is the mode actions are typically played in Photoshop's Actions Palette, unless logging is otherwise specified. In that case, Silent Mode is the mode of execution.

Silent Mode
Silent Mode allows you to completely automate regardless of states in Photoshop. If an error or other exceptional case arises while in execution of an action, the routine will pass back a descriptor with the error string under the key `keyMessage`. This string will be an explanation of the command that failed and why the command failed. It is up to you to deal with this legitimately. Note that any non-displayable error could come back this way. Typical operation stores all the strings in an error log specified by the user.

This mode allows for complete automation with no user interaction at all. It is, however, the most likely one to produce erroneous images because it will proceed with the the remaining events in the action on the current document even after a command has failed.

Elements, Classes, References, and Target Paths

We've discussed Events, Descriptor Blocks, Actions and Targets. Now, lets pull it all together. First, more definitions:

Action: An action is a set of one or more Photoshop events with target elements and parameters.

Event: A basic Photoshop command or instruction that is accessible by an automation plug-in. Events act on elements.

Element: An element consists of objects and classes and represents a tangible object inside Photoshop. Events act on elements and elements can be created, acted upon, saved in a file or destroyed. Elements can be simple objects (selections, colors, etc.) or they can be composed of lists of multiple objects with varied class parameters.

Class: A predefined set of Photoshop parameters that are logically connected to an object. For example, the classLayer has a set of subclasses including classTextLayer, classBackgroundLayer, classAdjustmentLayer, etc. The Rectangle class always consists of four numeric elements defining top, left, bottom and right parameters.

Target: The object of a Photoshop event.

Target Path: Also known as `keyTarget`. The relative path to a given selection. If no explicit path is given for a target in the action or event, Photoshop works its way up the containment hierarchy structure until it finds a suitable object for the action or event. For example, if the action is a blur on a selection on a channel, Photoshop looks for an element that recognizes a channel, then looks for the specified selection.

Reference: A target path through the Photoshop containment hierarchy.

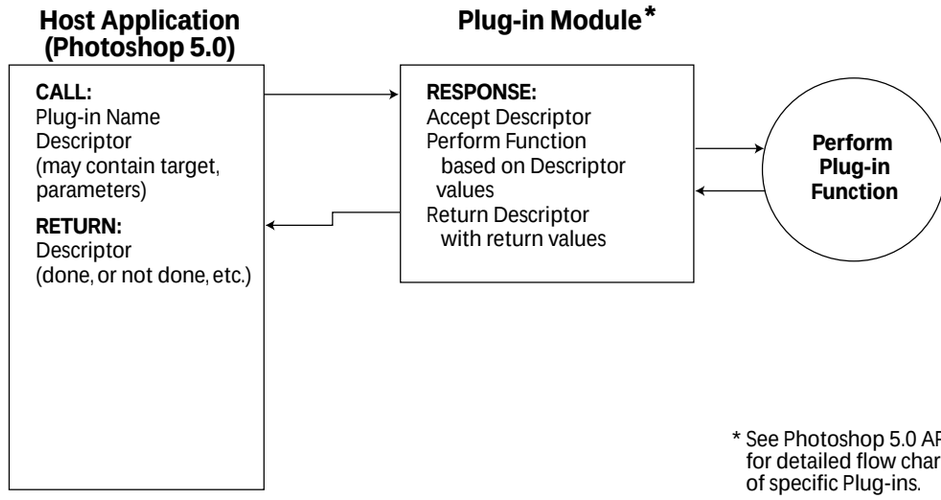
Most of the terminology in this chapter is high level definitions. Writing a new automation plug-in requires a much greater level of detail. The next chapter steps through the details of writing an automation plug-in.

3. Creating An Automation Plug-in

This chapter covers the basics of writing an automation plug-in and defines the various data structures and interrelationships between host, automation plug-ins and old style plug-ins.

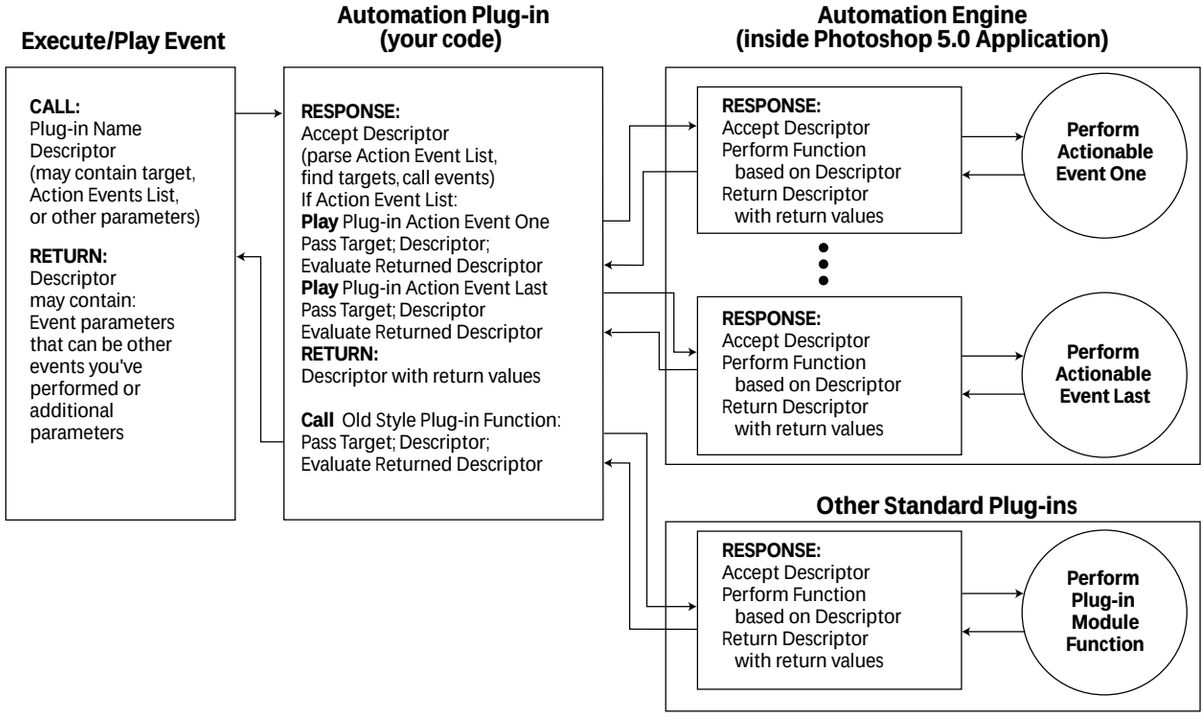
Standard Plug-ins vs. Automation Plug-ins

Old Style Photoshop Plug-in Module Functionality:
(single function, one call from host, information passed in descriptor block structure, no knowledge of other plug-ins)



* See Photoshop 5.0 API Guide for detailed flow charts of specific Plug-ins.

Automation Photoshop Plug-in Module Functionality:
(multifunction, called from host or other automation Plug-in, information passed in descriptor structure, contains calls to actionable events and standard plug-in functions)



The diagram above shows the differences between an old style Photoshop plug-in and an automation plug-in. The old style plug-in responds to a host command by performing its function and returning control to the host. In contrast, an automation plug-in is able to call internal Photoshop commands and external plug-ins via the Photoshop automation engine.

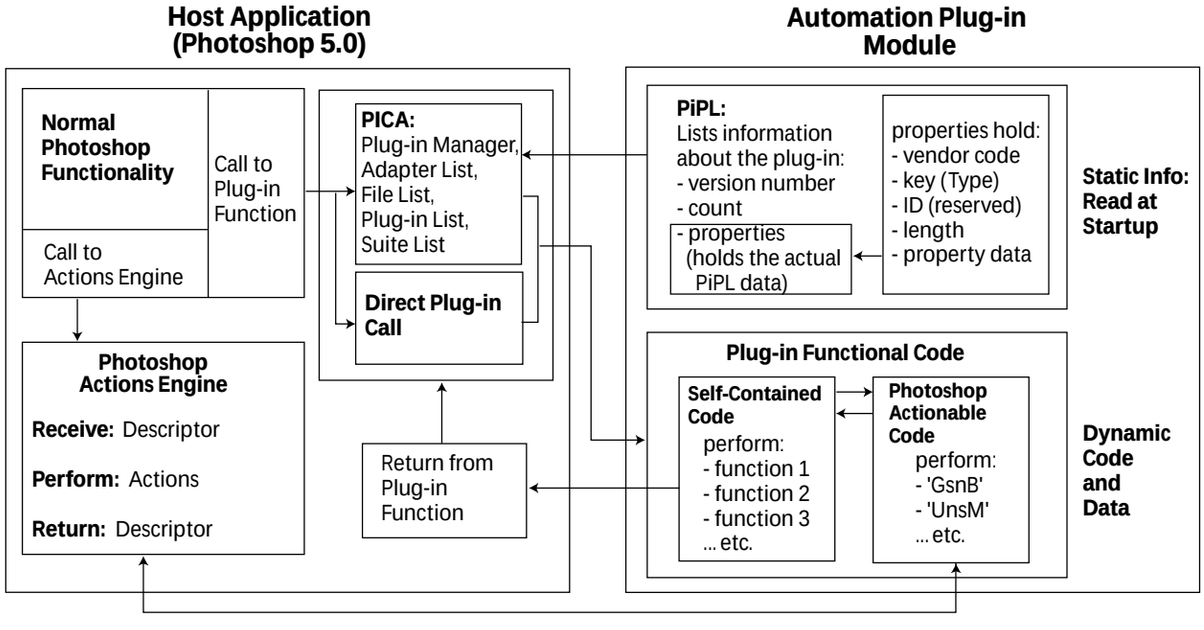
The differences between old style plug-ins and automation plug-ins can be summarized as follows: The old style Photoshop plug-in module is a self-contained functional block of code that receives a set of commands from Photoshop, does its function, and then returns control to Photoshop. It is completely self-contained, communicating with the host through a data block and callback functions. It cannot access any Photoshop actions or any other plug-in modules.

On the other hand, the automation plug-in module is a flexible block of code that can execute its own code as well as access any event. This code consists of a bundle of event codes and descriptors that decode into a set of actionable events. It calls the Photoshop actions event engine to perform those events, calls any other plug-in module as needed and returns a descriptor with the return values of the associated events. Unlike an old style Photoshop plug-in module, it can access any event through the Photoshop actions engine including external plug-in modules.

The Photoshop/Automation Plug-in Relationship

The key to mastering the Photoshop/automation plug-in relationship is understanding the data and code structures that are used to pass information between them. The key Photoshop elements are the application itself, PICA (Plug-In Component Architecture), direct Plug-in calls, and the Photoshop Actions Engine. On the Plug-in side, the key elements are the PiPL (Plug-in Properties List), any self-contained plug-in code and calls to the Photoshop Action Engine.

The Photoshop to Automation Plug-in Module Relationship:



Photoshop 5.0 Organization

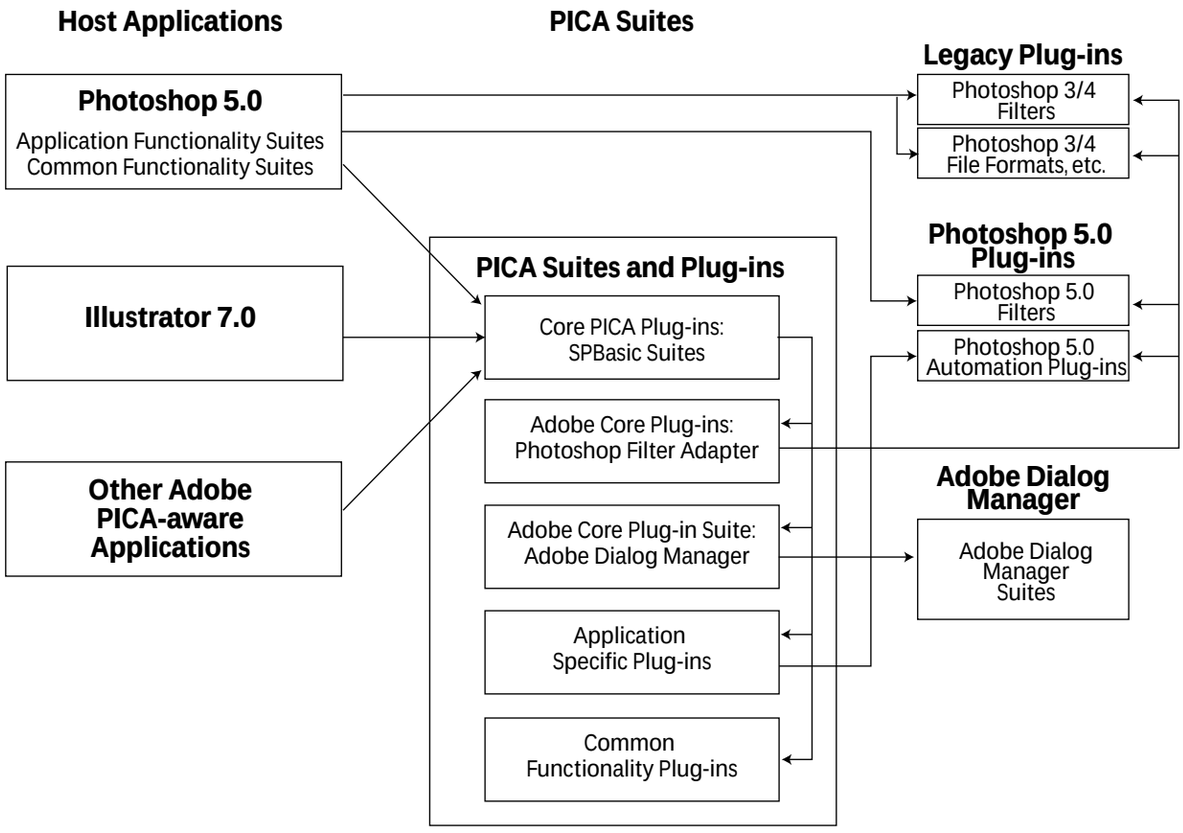
Photoshop's current internal organization reflects a combination of some historic and some new design elements. Starting with version 2, Photoshop pioneered the concept of a host/plug-in module architecture that spurred the development of many innovative third-party plug-ins. As Photoshop matured, the original plug-in architecture was revised to provide more robust structures. By Version 4, Photoshop itself was extensively revised internally to be more object oriented, and this restructuring was exploited for the user's benefit in the new Actions Palette. In Version 5 the automation plug-in architecture can now be exploited by external plug-ins. Photoshop includes the traditional code to access old style plug-ins as well as new ways to access the new automation Actions Engine.

The Automation Engine and Photoshop 5.0's actions hierarchical architecture are part of what enables the creation of automation plug-ins. The other part is the PICA suite-based plug-in architecture.

Plug-In Component Architecture (PICA)

PICA, the Plug-In Component Architecture, provides a way to communicate with internal and third-party plug-ins for several Adode applications. It is a robust and flexible method that allows hosts and automation plug-ins to communicate to a wide range of application commands and external plug-ins.

The PICA (Plug-In Component Architecture) Model



PICA allows the utilization of Adobe-created specific functionality plug-ins such as the cross-platform Adobe Dialog Manager (ADM). This useful PICA plug-in provides a means of creating dialog windows and boxes that

are easily portable across multiple platforms (i.e., you can write user interface code once).

NOTE: For more information, please refer to the *Adobe Plug-In Component Architecture PICA Reference* document contained in the Photoshop 5.0 SDK.

Plug-in Data Structures (PiPLs and Suites)

Key Automation plug-in data structures are the PiPL (Plug-in Properties List) and the message structures and suites used by PICA.

PiPLs

The PiPL is a static information resource in the plug-in itself. It contains information about the plug-in's function and its resources. It also contains the location of the entry point of the plug-in.

At start-up, the host searches the files located in Photoshop's Plug-In Folder and reads the PiPLs of any files found.

You must include in your PiPL: a vendorID code that identifies the host of the plug-in (in this case Photoshop); a propertyKey that specifies the type of plug-in (in this case, an automation plug-in); a propertyID that is (almost) always zero and can be considered reserved for future use; a propertyLength that contains the length of the propertyData field; and propertyData, a variable length field that contains the plug-in specific data. In this field, you'll put the `HasTerminology` structure that tells Photoshop that you have an `'aete'` dictionary and this structure includes ClassID, EventID, ResourceID, and a unique string.

PiPLs are documented in the *Cross-App Plug-In Resource Guide* document contained in the Photoshop 5.0 SDK. The `PiPL.r` file also contains a complete resource template for all Adobe PiPLs.

An example of a PiPL (the TriggerFilters plug-in PiPL) structure is shown below:

```
resource 'PiPL' (ResourceID, plugInName " PiPL", purgeable)
    {
        {
        Kind { Actions },
        Name { plugInName "..." },
        Category { "" },
        Version { (latestActionsPlugInVersion << 16) |
                    latestActionsPlugInSubVersion },

        #ifdef __PIMac__
            CodePowerPC { 0, 0, "" },
        #elif defined(__PIWin__)
            CodeWin32X86 { "ENTRYPOINT" },
        #endif

        HasTerminology
            {
            plugInClassID,
            plugInEventID,
            ResourceID,
```

```
            vendorName " " plugInName // Unique string.
            },
        EnableInfo { "true" },

        }
    };
```

Note that variables such as `plugInName`, `latestActionsPlugInVersion`, `latestActionsPlugInSubVersion`, `plugInClassID`, `plugInEventID`, `ResourceID`, `vendor name`, and `plugInName` are used for specific plug-in information so that these constants can be defined elsewhere once and used throughout your project. They can be easily updated as needed without affecting the code itself.

For `TriggerFilters` these variables are defined as follows:

```
#define plugInName            "TriggerFilters"
#define plugInCopyrightYear   "1998"
#define plugInDescription \
      "An example Actions Module to trigger filters in Adobe Photoshop®."
#define vendorName            "AdobeSDK"
#define plugInAETEComment     "triggerfilters example actions plug-in"
#define plugInSuiteID         'sdKE'
#define plugInClassID         plugInSuiteID
#define plugInEventID         'triG'
```

PiPLs are static data structures that define the basic properties of the plug-in. In contrast, suites are the active message passing structures.

Suites

Suites are structures defined by the host that specify the message structures and data organization used between plug-ins and Photoshop.

PICA defines a message passing architecture based on suite objects. PICA suites are acquired and released via two callbacks, AcquireSuite and ReleaseSuite. PICA is subdivided into a number of functional suites, starting with the SPBasic Suite. SPBasic acquires and releases the other suites and thus is the first thing called by any plug-in or application calling a PICA suite. The other suites perform such functions as acquiring and releasing plug-ins (as compared to suites), handling memory, file manipulations, calling non-PICA plug-ins, manipulating strings, and other housekeeping functions. PICA Suites are documented in the *Adobe Plug-In Component Architecture PICA Reference* document contained in the Photoshop 5.0 SDK.

PICA Suites are found in *SKD/SampleCode/Common/Headers/Photoshop/PICA* Folder.

For example, the SPBasic Suite is defined in `SPBasic.h` (found in *SDK/SampleCode/Common/Headers/Photoshop/SweetPea*) as follows:

```
typedef struct SPBasicSuite {

      SPAPI SPErr (*AcquireSuite)( char *name, long version, void **suite );
      SPAPI SPErr (*ReleaseSuite)( char *name, long version );
```

```
      SPAPI SPBoolean (*IsEqual)( char *token1, char *token2 );
      SPAPI SPErr (*AllocateBlock)( long size, void **block );
      SPAPI SPErr (*FreeBlock)( void *block );
      SPAPI SPErr (*ReallocateBlock)( void *block, long newSize,
                                                    void **newblock );
      SPAPI SPErr (*Undefined)( void );

} SPBasicSuite;


SPAPI SPErr SPBasicAcquireSuite( char *name, long version, void **suite );
SPAPI SPErr SPBasicReleaseSuite( char *name, long version );
SPAPI SPBoolean SPBasicIsEqual( char *token1, char *token2 );
SPAPI SPErr SPBasicAllocateBlock( long size, void **block );
SPAPI SPErr SPBasicFreeBlock( void *block );
SPAPI SPErr SPBasicReallocateBlock( void *block, long newSize,
                                                    void **newblock );
SPAPI SPErr SPBasicUndefined( void );
```

In addition to basic PICA suites, Adobe makes available the Adobe Dialog Manager or ADM suites. These suites simplify the process of developing cross-platform user interfaces. ADM is very useful since it provides many standard user interface functions such as tracking mouse movements, notifying on key presses, etc. This reduces the programming required for these types of user interface functions and provides the additional benefit of eliminating recoding of these functions for different platforms. ADM Suites are documented in the *Adobe Dialog Manager* document contained in the Photoshop 5.0 SDK.

NOTE: You must still initially create the actual dialog windows and elements for each platform. This can be done in any popular visual editor or in code.

In addition to PICA and ADM suites, Photoshop itself provides its own Photoshop specific suites that handle Photoshop functions. These are defined in the files found in *SDK/SampleCode/Common/Headers/Photoshop/Suites*.

```
AGMSPDraw.h                 PIMenuGroups.h
AGMSPPort.h                 PIMenuSuite.h
PIBufferSuite.h             PIProgressSuite.h
PIChannelPortsSuite.h       PIProxySuite.h
PIColorSpaceSuite.h         PIUIHooksSuite.h
PIErrorSuite.h              PIZStringsSuite.h
PIHandleSuite.h
```

NOTE: These folders and files are installed when you install the Photoshop 5.0 SDK.

Handling Suites In Your Plug-in

When creating your plug-in, you will use Photoshop suites (and possibly ADM suites) to perform various functions. A “suite dispatcher” is available in the `PIUBasic` library that automatically handles acquiring all of the most popular Photoshop suites. This code was created by Adobe Developer Relations and is included in an additional library project called `PIUBasic.mcp`. This project is included in the library folder of the

Automation SDK plug-in samples. It contains code and headers that perform much of the basic housekeeping functions you'll require.

In particular, `PIUSuites.cpp` (and `PIUSuites.h`) in `PIUBasic.mcp` acquires a subset (the main ones we think you'll need most) of the Photoshop suites for you. These are the main suites we think you'll need the most. You can always acquire other available suites on you own.

`PIUSuites.cpp` acquires and makes globally available the following suites:

Table 3–1: PIUSuites exported smart suite pointers

Suite Pointer	Suite Name
sSPBasic	PICA Basic suite
sSPRuntime	PICA Runtime utility suite.
sPSBasicActionControl	Photoshop Basic Action Control suite
sPSActionControl	Photoshop Action Control suite
sPSActionDescriptor	Photoshop Action Descriptor suite
sPSDescriptorRegistry	Photoshop Descriptor Registry suite
sPSActionList	Photoshop Action List suite
sPSActionReference	Photoshop Action Reference suite
sPSWindowNotify	Photoshop Macintosh Window dialog notify suite
sPSBuffer	Photoshop Buffer suite
sPSHandle	Photoshop Handle suite
sPSUIHooks	Photoshop User Interface utilities suite
sPSChannelPorts	Photoshop Channel Ports suite
sPSError	Photoshop Error Management Suite
sADMBasic	Adobe Dialog Manager basic functions uite
sADMDialog	Adobe Dialog Manager dialog functions suite
sADMNotify	Adobe Dialog Manager notification functions suite
sADMTrack	Adobe Dialog Manager tracking functions suite
sADMItem	Adobe Dialog Manager item functions suite
sADMList	Adobe Dialog Manager list functions suite
sADMEntry	Adobe Dialog Manager entry functions suite
sADMDialogGroup	Adobe Dialog Manager dialog group functions suite
sASZStrings	Adobe Systems Standard ZStrings suite

This is the subset of the total available Photoshop, PICA and ADM suites that you will most likely need.

When you write your plug-in and call the suite dispatcher (see the code in `PIUDispatch.cpp`), it will automatically check to see if the suites you want are available (loaded into memory), and give you an error message if any of the suites you need are not available.

The dispatch code will also determine what message has been sent to your plug-in (i.e., display your About Box, startup, shutdown, load into

memory, or unload from memory). Plug-ins only get loaded into memory when they are used. The dispatch code checks the parameters being passed to your plug-in, returns an error if there are any problems, and if not, jumps to your execute routine. If during execution, an error is returned, dispatch will handle normal errors (function done, return; or user canceled, return; etc.). This dispatch code may be used as is for the vast majority of tasks your plug-ins will perform.

The one requirement of using the dispatcher is you must identify which suites of the subset of Photoshop suites that the dispatcher automatically loads that you will NOT need. This is because the dispatcher assumes that you want access to all of the suites in its subset (see above list), but this will rarely be the case. In particular, automation plug-ins cannot perform operations on pixels; they can only call other plug-ins that do. Thus, any automation plug-in call to a pixel manipulation suite will result in an error.

For example, if an automation plug-in tries to access the ChannelPort suite, Photoshop won't let it have access to that suite. Thus, when the plug-in tries to access that suite, the pointer to the functions of that suite will be NULL. When the plug-in tries to use it, the dispatcher will realize that the suite is not available, it will display an error similar to "this suite is not available," and the plug-in will quit.

To avoid "not available" errors from suites you don't need, all you have to do is list the suites you don't require for your plug-in. The dispatcher will ignore those suites whether they are missing or not. An easy way to find out which suites you won't need is to run your plug-in. Any missing suites will generate an error dialog box that will display which suites are not available.

This can also help you catch any basic errors like forgetting to put the ADM plug-in into the Photoshop Plug-in Folder. For example, if the ADM plug-in is not in the Photoshop Plug-in Folder at startup and your plug-in uses ADM suites, Photoshop will return an error stating an ADM suite is not available and quit the plug-in.

Photoshop Suite Definitions

`PIActions.h` - contains Photoshop's suites related to automation. There are seven suites that Photoshop exports. One of them is the `PIActionDescriptor` suite that has forty functions. In addition to `PIActions.h`, there are some other header files in *samplecode/Common/Headers/Photoshop/PS-Suites* that contain Photoshop suites:

```
AGMSPDraw.h                PIErrorSuite.h       PIProgressSuite.h
AGMSPPort.h                PIHandleSuite.h      PIProxySuite.h
PIBufferSuite.h            PIMenuGroups.h       PIUIHooksSuite.h
PIChannelPortsSuite.h      PIMenuSuite.h        PIZStringsSuite.h
PIColorSpaceSuite.h
```

Adobe Developer Relations has added some additional utility functions into some suites. They are member functions associated with the suite pointer. To use them, you use the "." Reference operator: "`sPSActionDescriptor.SetReturnInfo(descriptor)`". The normal suite functions are available by double-dereferencing the suite:

```
sSPBasic->FreeBlock
          (
          Foo
          );
```

The PIActionDescriptor suite as defined in `PIActions.h` is shown below:

```
#define kPSActionDescriptorSuite"df135115-c769-11d0-8079-00c04fd7ec47"
#definekPSActionDescriptorSuiteVersion2

typedef struct PSActionDescriptorProcs
{
       // ALLOCATES: descriptor
       SPAPI OSErr (*Make)(PIActionDescriptor *descriptor);

       SPAPI OSErr (*Free)(PIActionDescriptor descriptor);

       SPAPI OSErr (*GetType)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              DescriptorTypeID *type);

       // index is zero based
       SPAPI OSErr (*GetKey)(PIActionDescriptor descriptor, uint32 index,
                              DescriptorKeyID *key);

       SPAPI OSErr (*HasKey)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              Boolean *hasKey);
       SPAPI OSErr (*GetCount)(PIActionDescriptor descriptor, uint32 *count);
       SPAPI OSErr (*IsEqual)(PIActionDescriptor descriptor, PIActionDescriptor other,
                              Boolean *isEqual);

       SPAPI OSErr (*Erase)(PIActionDescriptor descriptor, DescriptorKeyID key);
       SPAPI OSErr (*Clear)(PIActionDescriptor descriptor);

       SPAPI OSErr (*PutInteger)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              int32 value);
       SPAPI OSErr (*PutFloat)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              double value);
       SPAPI OSErr (*PutUnitFloat)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              DescriptorUnitID unit, double value);
       SPAPI OSErr (*PutString)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              char *cstrValue);
       SPAPI OSErr (*PutBoolean)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              Boolean value);
       SPAPI OSErr (*PutList)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              PIActionList value);
       SPAPI OSErr (*PutObject)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              DescriptorClassID type,  PIActionDescriptor value);
       SPAPI OSErr (*PutGlobalObject)(PIActionDescriptor descriptor,
                              DescriptorKeyID key, DescriptorClassID type,
                              PIActionDescriptor value);
       SPAPI OSErr (*PutEnumerated)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              DescriptorEnumTypeID type, DescriptorEnumID value);
       SPAPI OSErr (*PutReference)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              PIActionReference value);
       SPAPI OSErr (*PutClass)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              DescriptorClassID value);
       SPAPI OSErr (*PutGlobalClass)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              DescriptorClassID value);
       SPAPI OSErr (*PutAlias)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              Handle value);

       SPAPI OSErr (*GetInteger)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              int32* value);
       SPAPI OSErr (*GetFloat)(PIActionDescriptor descriptor, DescriptorKeyID key,
                              double* value);
       SPAPI OSErr (*GetUnitFloat)(PIActionDescriptor descriptor, DescriptorKeyID key,
```

```
                                DescriptorUnitID* unit, double* value);
        SPAPI OSErr (*GetStringLength)(PIActionDescriptor descriptor,
                                DescriptorKeyID key, uint32 *stringLength);
        SPAPI OSErr (*GetString)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                char *cstrValue, uint32 maxLength);
        SPAPI OSErr (*GetBoolean)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                Boolean* value);
        SPAPI OSErr (*GetList)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                PIActionList* value);
        SPAPI OSErr (*GetObject)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                DescriptorClassID* type, PIActionDescriptor* value);
        SPAPI OSErr (*GetGlobalObject)(PIActionDescriptor descriptor,
                                DescriptorKeyID key, DescriptorClassID* type,
                                PIActionDescriptor* value);
        SPAPI OSErr (*GetEnumerated)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                DescriptorEnumTypeID* type, DescriptorEnumID* value);
        SPAPI OSErr (*GetReference)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                PIActionReference* value);
        SPAPI OSErr (*GetClass)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                DescriptorClassID* value);
        SPAPI OSErr (*GetGlobalClass)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                DescriptorClassID* value);
        SPAPI OSErr (*GetAlias)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                Handle* value);

        // Convienence Functions

        SPAPI OSErr (*HasKeys)(PIActionDescriptor descriptor, DescriptorKeyIDArray
                                requiredKeys, Boolean *hasKeys);
        SPAPI OSErr (*PutIntegers)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                uint32 count,  int32* value);
        SPAPI OSErr (*GetIntegers)(PIActionDescriptor descriptor, DescriptorKeyID key,
                                uint32 count, int32* value);

        SPAPI OSErr (*AsHandle)(PIActionDescriptor descriptor,
                                PIDescriptorHandle *value);

} PSActionDescriptorProcs;
```

To access this suite, use `sPSActionDescriptor`, which is defined in PIUSuites.h. For example, `sPSActionDescriptor->Put Boolean (descriptor, key, boolean value);`

The Plug-in Development Process

As detailed in the *Graphics and Publishing Software Development Toolkit Technical Notes FAQ: Frequently Asked Questions:* the Plug-in Development Process should following this sequence.

Table 3–2: Plug-in Development Process

1.	Use the application you're interested in making the plug-in for (in this case Photoshop).
2.	Do a thorough search and use the plug-ins already on the market.
3.	If you still see a need for your proposed plug-in:
4.	Get the Photoshop SDK.
5.	READ THE DOCUMENTATION.
6	Read the header files.
7.	Compile the sample code.
8.	Formulate your approach based on modifying a sample that already works and ships with the SDK. (We encourage this.)
9.	GO BACK AND READ THE DOCUMENTATION, now that you know what it is you want to do.
10.	Start modifying an existing, working sample from the SDK.
11.	If you encounter some problems, backtrack all the way back to the sample that already works. You did start with a working sample, didn't you?
12	If you still encounter some problems, refer to the header files.

Table 3–2: Plug-in Development Process

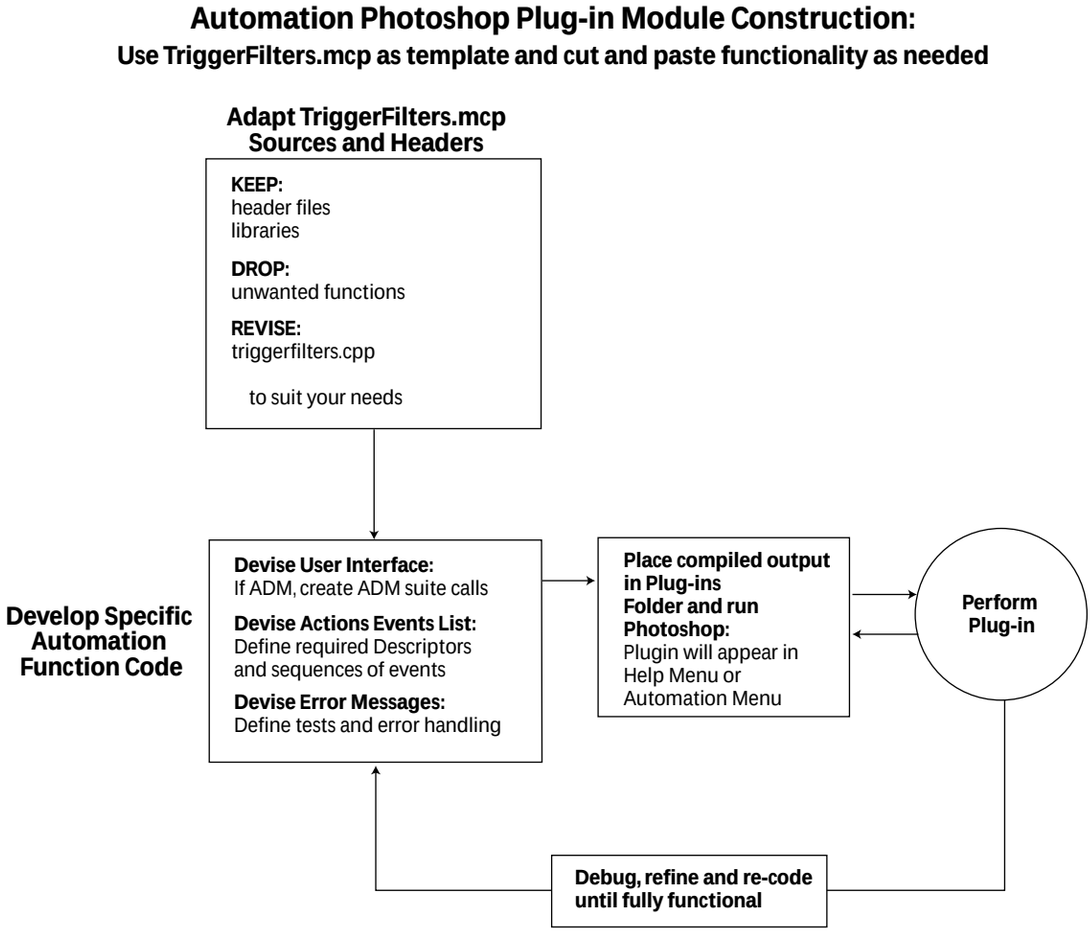
13.	If you still encounter some problems, refer to the documentation.
14.	If you still encounter some problems, QA your plug-in and determine exactly where its crashing.
15.	If you still encounter some problems, e-mail all this information to us. Help us help you by doing stringent QA on your code before you talk to us. We cannot write your code for you, and we're not responsible for QAing your plug-ins.
16.	Finish your plug-in.
17.	Write documentation.
18.	E-mail us and arrange an appointment with our evangelism and marketing teams to talk about your marketing strategy.
19.	E-mail us and arrange with our engineering teams to get your plug-in into our QA department to QA with the latest build of our applications.
20.	Market your plug-in.
21.	Go to step 1 with another Adobe application.

When creating your **automation** plug-in, there are several basic technical steps to follow. The first thing is to consider what functions your plug-in will execute and what parameters will be associated with those functions. For example, to create a new automation plug-in that creates a new document (a simple, but basic function), there are four basic parameters to consider: the width, the height, the resolution of the image and the color mode of the file. Thus, your plug-in will have four parameters: three of the type integer (W, H, Res) and one of the type enumerated (color mode).

To aid in the development of automation plug-ins, Adobe has provided some sample automation plug-ins such as the `TriggerFilters` and the `MakeNew` plug-ins as CodeWarrior projects on the Macintosh and Visual C++ projects on Windows. In the projects are source files, resource files, header files, and library files that you can adapt to your own requirements.

First, find the sample automation plug-in closest to your needs and copy the project folder, including headers and code files. Then rename all the files the same as your plug-in's name and begin the process of revising the files to contain your plug-in's functionality.

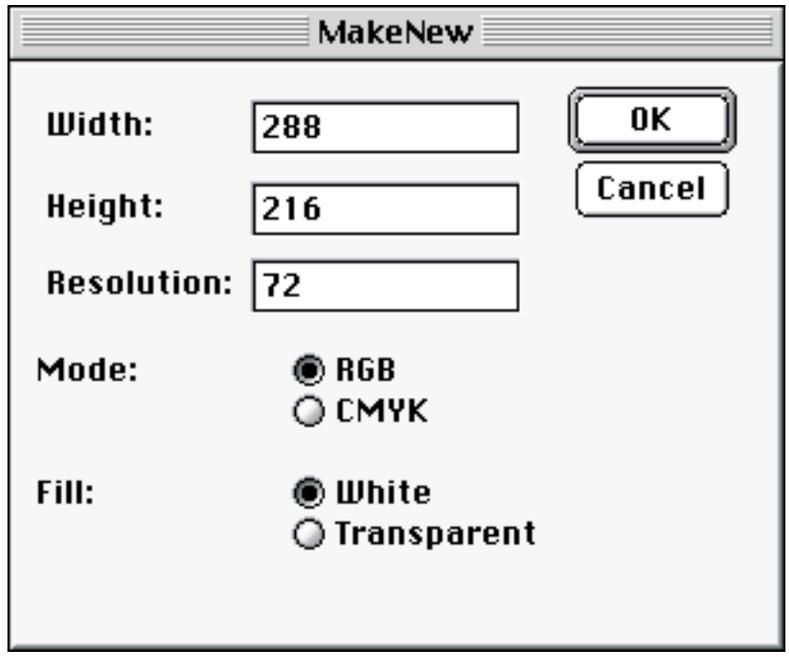
The next figure illustrates this process.



Our new automation plug-in

In this document we will create a automation plug-in named `MakeNew`. It will be a simple one, but it will illustrate the process you can use to create more complex automation plug-ins.

`MakeNew` will be called from the Automate menu and will perform a simple function: create a new document with user supplied parameters. Its user interface is shown below:



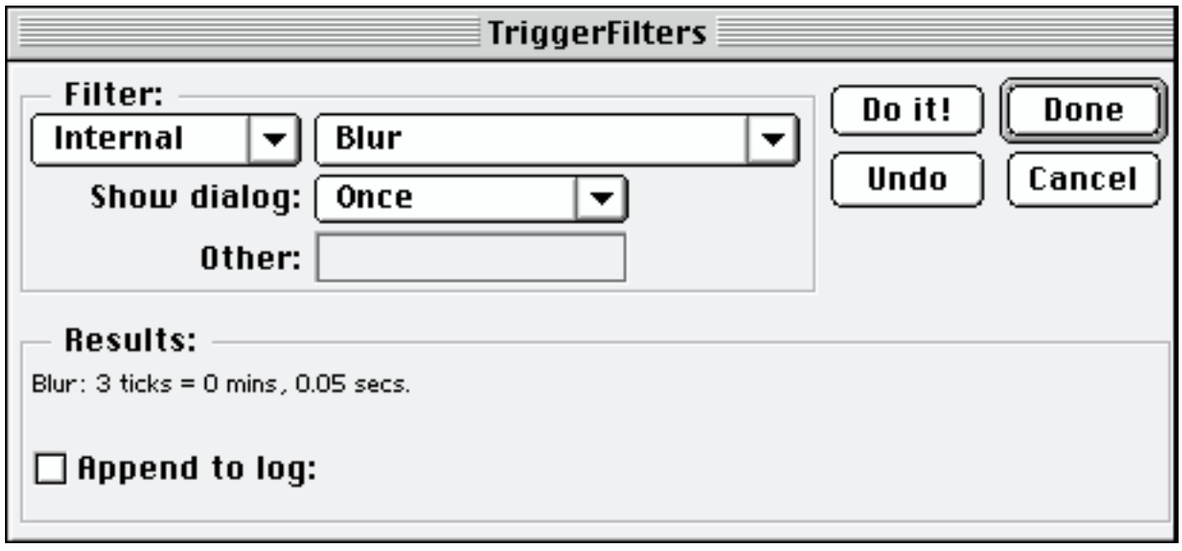
Default values for width, height, resolution, mode and fill are supplied as shown.

`MakeNew` incorporates all of the basic structures you'll need to create more powerful plug-ins. In creating `MakeNew`, we'll cover all of the automation source and header files and we'll detail creating dialog, PiPL, and `'aete'` resources in this document.

We'll create `MakeNew` by taking an existing automation plug-in, `TriggerFilters.8li` and revising it to suit our needs.

Examining the TriggerFilters plug-in project

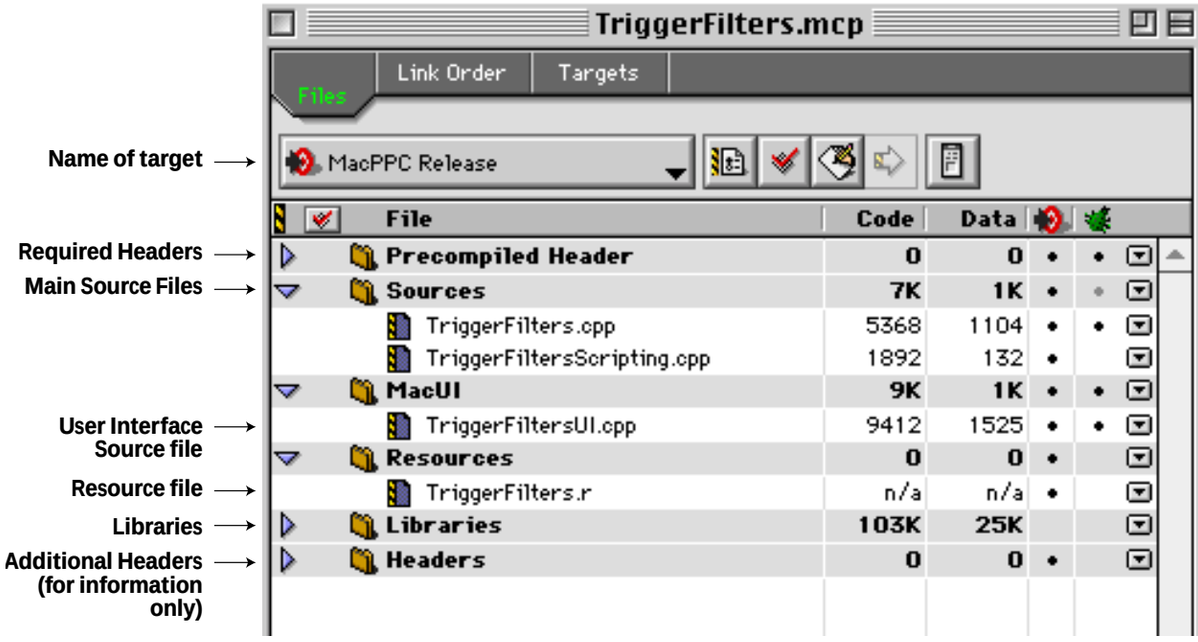
`TriggerFilters` is a fairly complex plug-in that uses ADM to manipulate a dialog box with various options. As shown in the figure below, the user can select a specific filter, select the dialog options (never display, display once or display aways), run a selected filter and display the timing results. Optionally, the user may log results in a text file.



The figure below shows the TriggerFilters project files from the Photoshop 5.0 SDK as shown in the CodeWarrior IDE 2.1 environment.

The key elements are the target (the chosen platform), in this case the Macintosh, header files, source files, resource files, and libraries.

Photoshop Automation Plug-in Make Files:
(Sources and Resource files/
CodeWarrior Power Mac shown)



Precompiled headers are required, as are the source, resource, and library files. For information purposes and easy access, additional headers files are included in the headers folder.

There are four types of files: headers, sources, resources and libraries. There are several header files:

`TriggerFilters.pch`,
`TriggerFiltersTerminology.h`, and
`TriggerFilters.h`.

There are three source files:

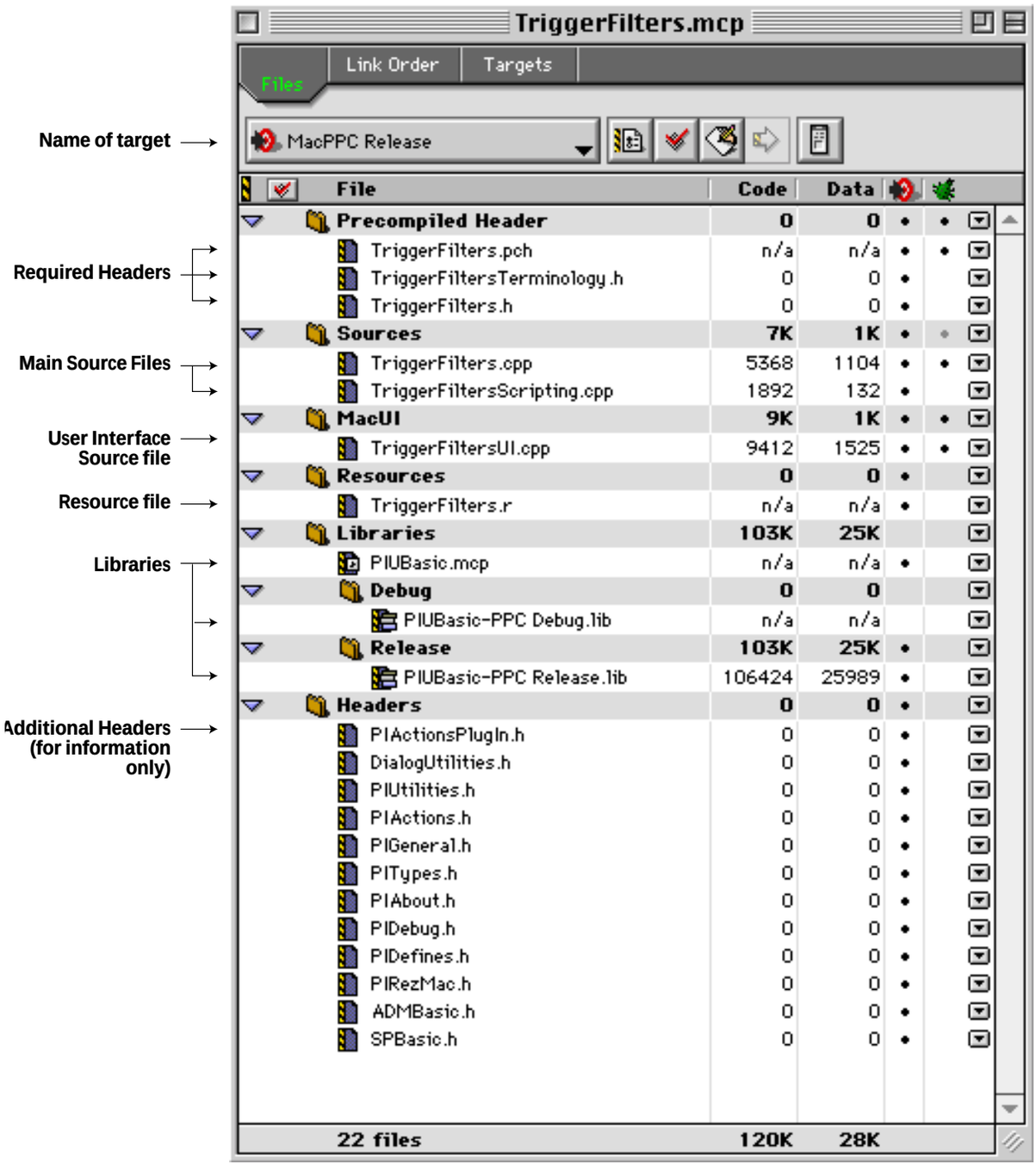
`TriggerFilters.cpp`,
`TriggerFiltersScripting.cpp`, and
`TriggerFiltersUI.cpp`.

There is one resource file: `TriggerFilters.r`.

Finally, there are several library files:`PIUBasic.mcp` is an entire CodeWarrior project with its own source, headers, resource and libraries; and there are debug and release library files.

The Additional Headers group contains useful files for quick access to various Photoshop and automation functions.

Photoshop Automation Plug-in Make Files:
(CodeWarrior Power Mac Shown)



The main source file `TriggerFilters.cpp` includes basic include files (the headers and precompiled headers), the global variable definitions required for `TriggerFilters`, the prototypes for EntryPoint (the main function in this file), the actual EntryPoint code, the dispatch code (see above descriptions) and code for each of the functions required for `TriggerFilters`. These elements are shown in the following diagram. The second source file, `TriggerFiltersScripting.cpp`, handles reading and writing descriptors. It uses `ReadScriptParams()` and `WriteScriptParams()` to accomplish these automation functions. The third source file, `TriggerFiltersUI.cpp` handles the user interface functions. The `TriggerFilters.r` resource file contains the actual dialog resources as well as the PiPL and `'aete'` dictionary resources.

This diagram shows the source and header files of the `TriggerFilters` project with a brief listing of what's inside each source file.

Trigger Filters Plug-in Module Files:

TriggerFilters.cpp (C++ source file)

#includes:
PIDefines.h
TriggerFilters-PPC.ch or TriggerFilters.h

globals:
```
char gOtherEvent[kMaxStr255Len] = "";
bool gSuppressDialog = false;
bool gShowDialog = false;
bool gLogToFile = false;
SPPlatformFileSpecification gFile;
eventList_t* gLastEventList  = NULL;
eventList_t* gEventList   = NULL;
size_t gEventList_size = 0;
```

prototypes:
```
SPAPI SPErr ENTRYPOINT
 (
 const char* const caller,   // Area that's calling us.
 const char* const selector,  // Specific action to perform.
 const void* const data   // Data related to command.
 );
```

ENTRYPOINT /main:
```
SPAPI SPErr ENTRYPOINT
 (
 const char* const caller,
 const char* const selector,
 const void* const data
 )
 {
 SPErr error = kSPNoError;
```

dispatching:
```
static PIUDispatch* dispatcher = new PIUDispatch
  (
  AboutID,   // Your About string ID or 0.
  PIUAPI_None,  // Your about routine or PIUAPI_None.
  Execute,  // Your execute routine or PIUAPI_None.
  PIUAPI_None,  // Your Reload or PIUAPI_None.
  PIUAPI_None,  // Your Unload or PIUAPI_None.
  PIUAPI_None,  // Your Startup or PIUAPI_None.
  PIUAPI_None   // Your Shutdown or PIUAPI_None.
  );
 error = dispatcher->Dispatch(caller, selector, data);
 if (dispatcher->Done() || error != kSPNoError)
 {
  delete dispatcher;
 }
 return error;
 }
```

EXECUTE TriggerFilters CODE:

SPErr Execute(void)
Make Filter List
Free Filter List
Do Undo
Play Filter
Find ID in Event List
Find Name in Event List
Find Index in Event List
Append to File

Header Files:

Required Precompiled Headers:
TriggerFilters.pch
TriggerFiltersTerminology.h
TriggerFilters.h or TriggerFilters-PPC.ch

Additional Headers (for information only):

PIActionsPlugin.h	PIAbout.h
DialogUtilities.h	PIDebug.h
PIUtilities.h	PIDefines.h
PIActions.h	PIRezMac.h
PIGeneral.h	ADMBasic.h
PITypes.h	SPBasic.h

TriggerFiltersScripting.cpp (C++ source file)

#includes:
PIDefines.h
TriggerFilters-PPC.ch or TriggerFilters.h

Read Scripting Parameters:
```
SPErr ReadScriptParams ()
{
 SPErr error = kSPNoError;
 if ... various tests ....  then ...
```

Write Scripting Parameters:
```
SPErr WriteScriptParams ()
{
 SPErr    error = kSPNoError;
 if ... various tests .... then ...
```

TriggerFiltersUI.cpp (C++ source file)

#includes:
TriggerFilters-PPC.ch or TriggerFilters.h

Definitions:
dialog box enumerations

prototypes:
DoUI, DoDialogCleanup, InitDialogList, etc.

globals:
```
bool gCancelButtonIsReset= false;
groupList_t*       saveLastGroupList =NULL;
geventList_t*      saveLastEventList=NULL;
bool saveLogToFile = false; etc.
```

EXECUTE UI CODE:
DoUI
DoUIInit
DoDialogCleanup, etc.

Other Resources

Resources
TriggerFilters.r

Libraries:
PIUBasic.mcp
PIUBasic-PPC Debug.lib
PIUBasic-PPC Release.lib

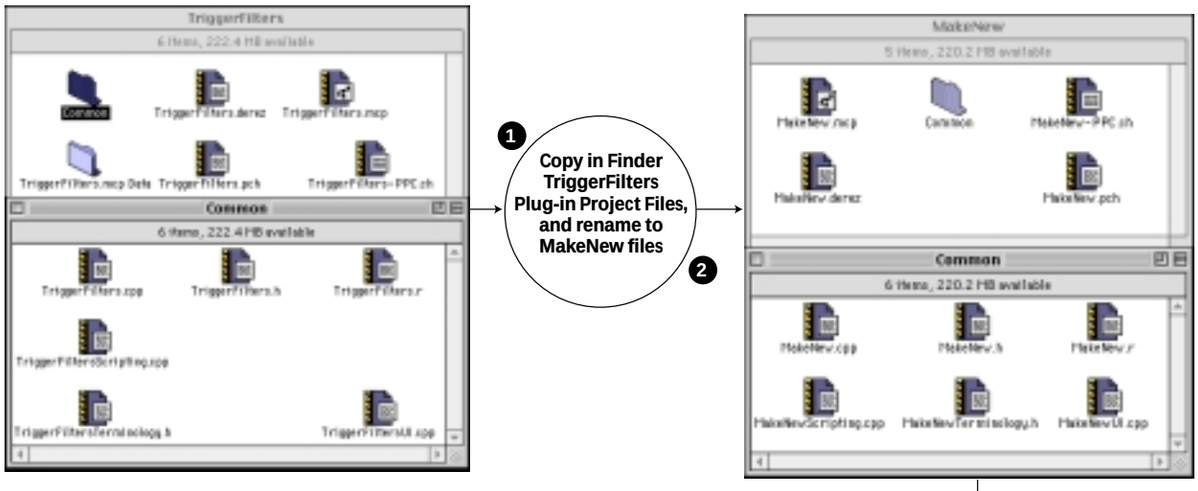
The final piece is the `PIUBasic.mcp` project, included in the library folder. It contains code and headers providing much of the basic housekeeping functions required by any plug-in. See Appendix B for more information.

Writing Your Own Plug-in: Step-by-Step

The fastest way to create your own Photoshop automation plug-in is to copy the project files from an existing automation plug-in and revise them to suit your needs. We'll explain how we created the `MakeNew.8li` plug-in from the `TriggerFilters.8li` plug-in.

The next three diagrams show the process, and the following text explains each step. Note that the finished `MakeNew` project is included in your SDK files as well as the `TriggerFilters` project.

MakeNew Plug-in Module Step-by-step Construction Part 1:
Copy TriggerFilters.mcp files and cut and paste

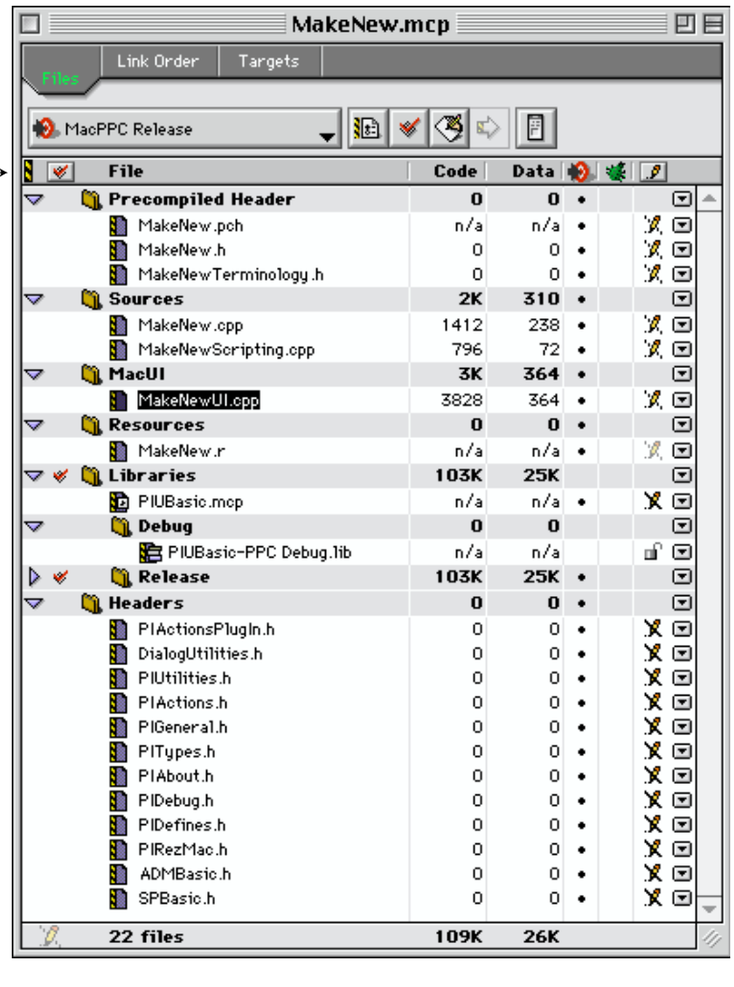


Determine which files you'll keep

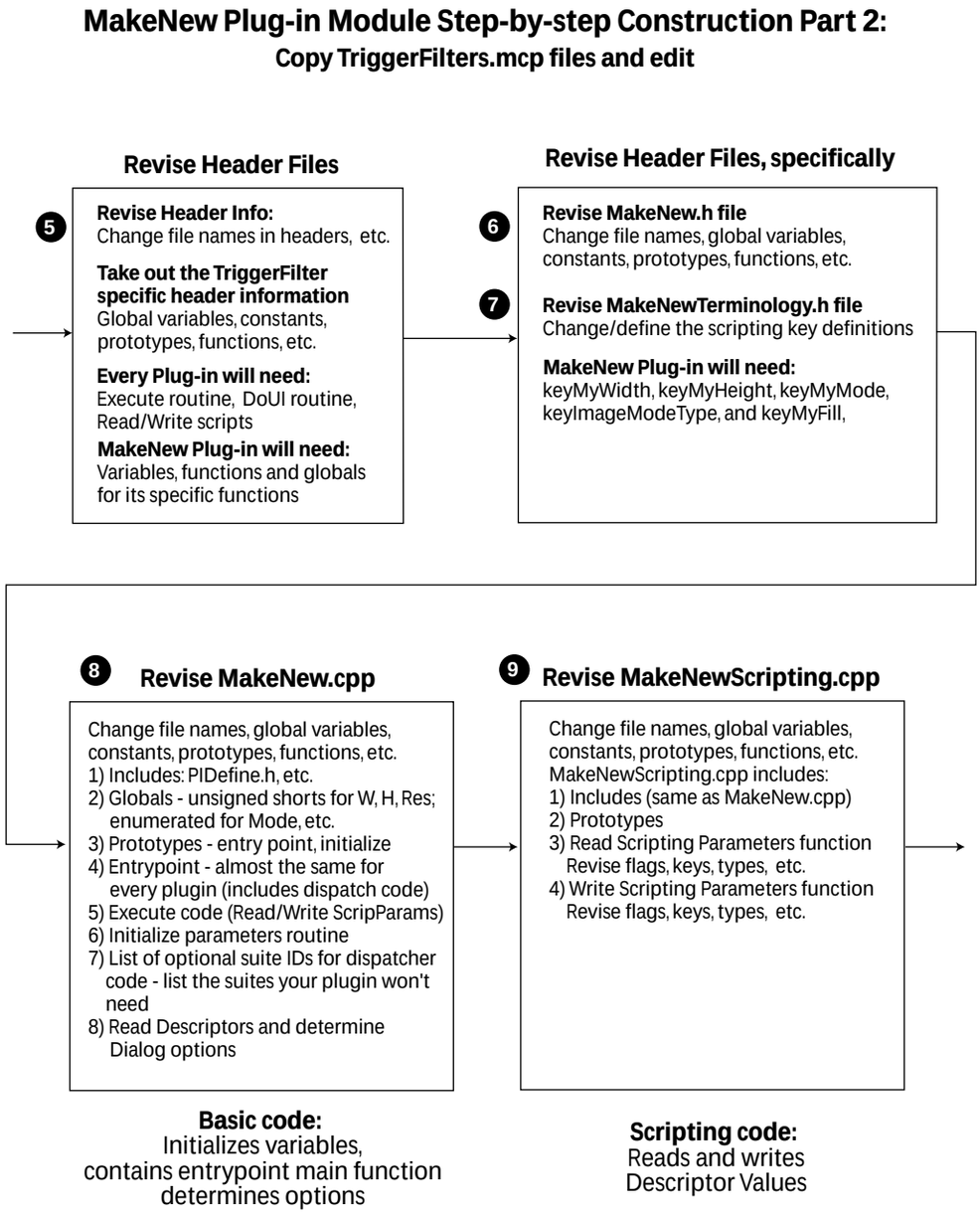
Keep all of the header and library files

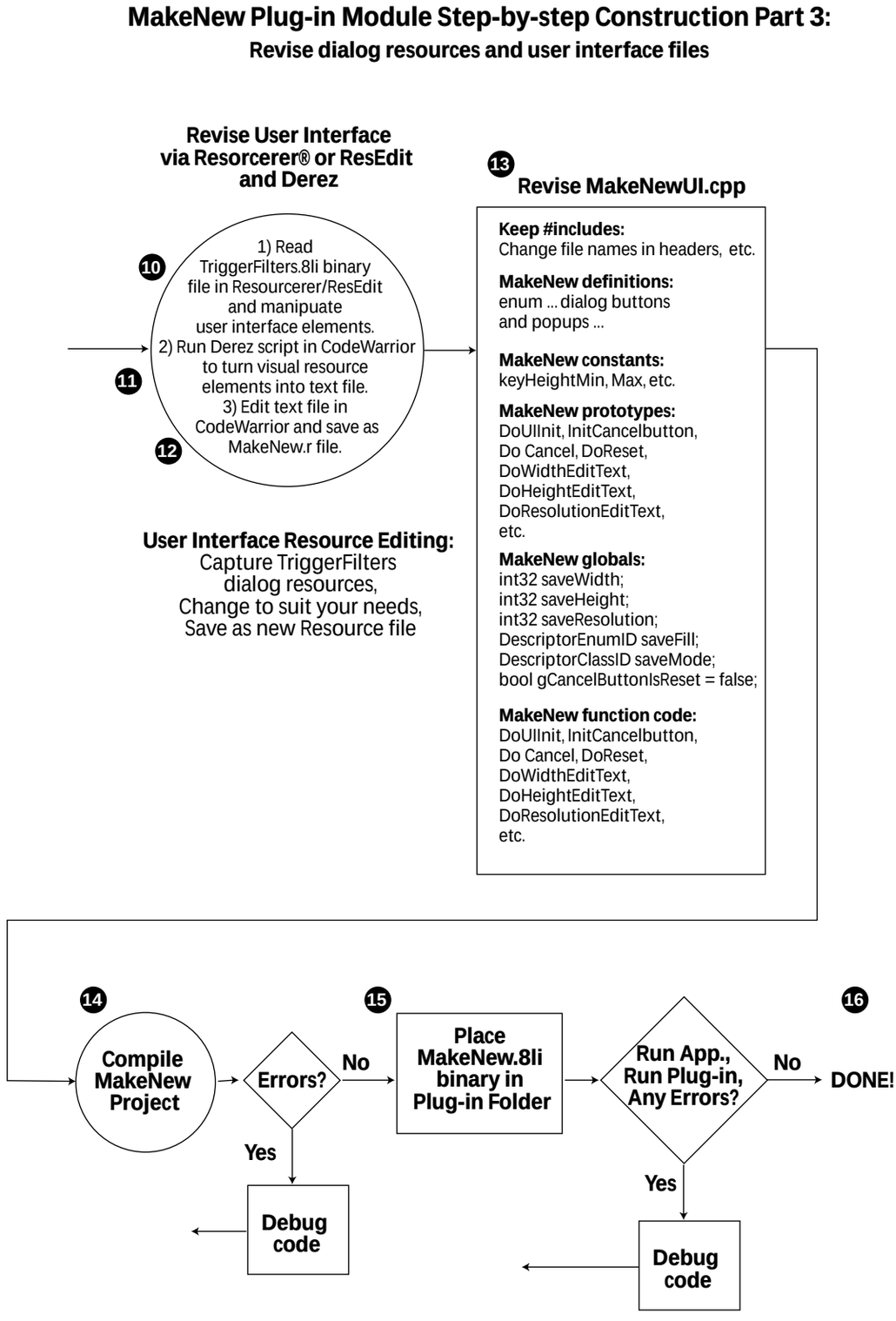
3 Add newly renamed files to MakeNew project and delete the TriggerFilter files

4 Keep the project target settings, but change the target file name to: MakeNew.8li



In this example, a new automation plug-in, MakeNew will be created using TriggerFilters as a template. MakeNew will be a simple automation plug-in that will open a user dialog box and based on user input on Width, Height, Resolution, Fill and Color Mode, let the user create a new document. This simple automation plug-in will illustrate the process and show what elements are absolutely needed for an automation plug-in.





Step 1 - Copy the `TriggerFilters.mcp` project folder.

Copy the project files from an existing automation plug-in into a new directory called “Your Plug-in Name.” In this case, we are creating a new plug-in called MakeNew, so the new project name is `MakeNew.mcp`.

Step 2 - Rename all files.

Rename all project files, replacing `TriggerFilters` with `YourName` (in this case `MakeNew`). Delete the `MakeNew.mcp Data` folder, since this information will be recreated when we compile our new project.

Step 3 - Open the newly named `MakeNew.mcp` project. Inside will be all of the old TriggerFilters files. Use Add Files and insert all of the renamed

MakeNew files. Add them in the same folders as their TriggerFilters counterparts (i.e., source files in the “source” folder, headers in the “header” folder, etc.). Once the new files are inserted, delete the old TriggerFilter files.

Step 4 - In the project settings change the target output file to `MakeNew.8li` for both the Debug and Release Targets.

Step 5 - Revise the Header file information to take out any TriggerFilters information and replace with MakeNew information.

Step 6 - Open `MakeNew.h` and replace the TriggerFilters information with MakeNew specifics. Replace the TriggerFilters globals with the MakeNew global variables. For MakeNew, these variables are:

```
extern int32 gWidth;
extern int32 gHeight;
extern int32 gResolution;
extern DescriptorEnumID gFill;
extern DescriptorClassID gMode;
```

These are the variables for the width, height, resolution, Fill mode and Color Mode.

MakeNew has no constant definitions, so the TriggerFilter constant definitions may be removed and not replaced.

MakeNew requires the following global prototypes: DoUI, ReadScriptParams, and WriteScriptParams. These exist in TriggerFilters (along with several others), so delete everything but these three prototypes:

```
SPErr DoUI ();                                  // Show the UI.

SPErr ReadScriptParams ();                      // Read any scripting params.
SPErr WriteScriptParams ();                     // Write any scripting params.
```

These functions will return an `SPErr` value that will be `kSPNoError` if all is well, or a specific error code if there is a problem.

Step 7 - Revise the `MakeNewTerminology.h` file to include the MakeNew specific keys and their definitions. The definitions will be the scripting keys for the new document variables (Width, Height, Resolution, Fill, Color Mode):

```
#define keyMyWidth              keyWidth
#define keyMyHeight             keyHeight
#define keyMyResolution         keyResolution
#define keyMyFill               keyFill
#define typeMyFill              typeFill
#define keyMyMode               keyMode // RGB, CMYK, etc.
#define typeClassMyMode         typeClassMode
```

Because Fill is an enumerated value, it requires a type key (`typeMyFill`) in addition to a Fill key. Because color mode is actually a color class in

Photoshop, it is a class type and requires a key mode (`keyMyMode`) and a class mode type (`typeClassMyMode`).

In addition, two reset and cancel string ID resource keys are defined so that the Cancel button will toggle in the UI:

```
#define kResetStringID        uiID
#define kCancelStringID       kResetStringID+1
```

uiID is predefined as 16001.

Step 8 - Revise the `MakeNew.cpp` source file. This is our main source file, and it contains the four functions that we need to create a new document. It is organized exactly like `TriggerFilters.cpp` (see earlier diagram) with just the MakeNew elements. We need to have our includes:

```
#include "PIDefines.h"

#if defined(__PIMWCWMacPPC__)
      #include "MakeNew-PPC.ch"
#else
      #include "MakeNew.h"
#endif
```

and our global variables:

```
int32 gWidth;
int32 gHeight;
int32 gResolution;
DescriptorEnumID gFill; // white, transparent, background
DescriptorClassID gMode; // RGB, CMYK, etc.
```

and our prototypes for `ENTRYPOINT`, `Initialize`, `ConvertToUnitDistance` and `MakeNewDocument` functions:

```
SPAPI SPErr ENTRYPOINT
      (
      const char* const caller,     // Area that's calling us.
      const char* const selector,   // Specific action to perform.
      const void* const data        // Data related to command.
      );

// Initialize our parameters:
static void Initialize (void);

// Convert from any base to unitDistance:
static double ConvertToUnitDistance
      (
      const double amount,
      const double amountBasePerInch
      );

// Actually make a new document:
static SPErr MakeNewDocument (void);
```

`ENTRYPOINT` is the first routine in the source file. It is called first when the plug-in is called. `Initialize` is a function that initializes our variables (Width, Height, etc.). `ConvertToUnitDistance` is a small utility function that we've added that converts a distance into standard `unitDistance` units of 72 dpi. This lets the user input a parameter for Width or Height in pixels and our plug-in automatically converts it into distance. In other

words, if the user inputs the values of 288 pixels for Width, 216 pixels for Height, and 72 dpi for Resolution, our plug-in will create an image 4 inches wide and 3 inches high with a resolution of 72 dpi.

The `MakeNewDocument` actually performs our plug-in task of making a new document. This is where the plug-in accesses the Photoshop Automation Engine. Here is the prototype:

```
static SPErr MakeNewDocument (void);
```

After the function prototypes, we put the actual code for our four functions, starting with `ENTRYPOINT`. The entrypoint will be “pascal void” for Macintosh programs and “void” for Windows.

```
SPAPI SPErr ENTRYPOINT
    (
    const char* const caller,
    const char* const selector,
    const void* const data
    )
    {
    SPErr error = kSPNoError;

    // This is the list of suites we will consider "optional".  The rest are
    // loaded automatically and will generate an error if not available:
    const char* optionalSuiteIDs [] =
        {
        kPSBasicActionControlSuite,
        kPSChannelPortsSuite,
        NULL// Make sure this list ends with NULL.
        };

    // This C++ object takes care of all dispatching.
    static PIUDispatch* dispatcher = new PIUDispatch
        (
        (char**) &optionalSuiteIDs,//List of optional suites.

        AboutID,         // Your About string ID or 0.
        PIUAPI_None,     // Your about routine or PIUAPI_None.

        Execute,         // Your execute routine or PIUAPI_None.

        PIUAPI_None,     // Your Reload or PIUAPI_None.
        PIUAPI_None,     // Your Unload or PIUAPI_None.

        PIUAPI_None,     // Your Startup or PIUAPI_None.
        PIUAPI_None      // Your Shutdown or PIUAPI_None.
        );

    error = dispatcher->Dispatch(caller, selector, data);

    if (dispatcher->Done())
        {
        delete dispatcher;
        }

    return error;
    }
```

ENTRYPOINT uses the PICA messaging, with the "caller, selector, data" format. Then, there is the list of optional suites, ending in NULL. As explained earlier in the "Handling Suites in Your Plug-in" section, PIUSuites.cpp (and PIUSuites.h) in PIUBasic.mcp acquires a subset (the main ones we think you'll need most) of the Photoshop suites for your plug-in. You are required to list the plug-ins from this subset that you will NOT need. Here we list:

```
const char* optionalSuiteIDs [] =
    {
    kPSBasicActionControlSuite,
    kPSChannelPortsSuite,
    NULL    // Make sure this list ends with NULL.
    };
```

This says that we will not need the PSBasicActionControlSuite and the PSChannelPortSuite for our MakeNew plug-in.

Then, we put in the dispatch code (previously mentioned in the "Handling Suites in Your Plug-in" section of this chapter) as is from the TriggerFilters.cpp file.

```
static PIUDispatch* dispatcher = new PIUDispatch
    (
    (char**) &optionalSuiteIDs,// List of optional suites.

    AboutID,         // Your About string ID or 0.
    PIUAPI_None,     // Your about routine or PIUAPI_None.

    Execute,         // Your execute routine or PIUAPI_None.

    PIUAPI_None,     // Your Reload or PIUAPI_None.
    PIUAPI_None,     // Your Unload or PIUAPI_None.

    PIUAPI_None,     // Your Startup or PIUAPI_None.
    PIUAPI_None      // Your Shutdown or PIUAPI_None.
    );

error = dispatcher->Dispatch(caller, selector, data);

if (dispatcher->Done())
    {
    delete dispatcher;
    }

return error;
```

This dispatcher C++ object takes care of all of the loading and unloading of your plug-in, as well as any startup, shutdown, and About Box calls.

PIUDispatcher will handle all of the basic housekeeping functions involved in calling your plug-in (load, reload, etc.). Your Execute function should read incoming scripting parameters that tell your plug-in what function to perform. Your automation plug-in should perform according to the descriptor handed to it. In the sample, descriptors are read using ReadScriptParams().

```
SPErr Execute(void)
    {
    SPErr error = kSPNoError;

    Initialize(); // Default parameters for all globals.

    ReadScriptParams(); // Override globals with new descriptor info.

    // Determine if we need to pop our dialog:
    PIDialogPlayOptions playInfo = plugInDialogDisplay;
    sPSActionDescriptor.GetPlayInfo(NULL, &playInfo);

    // Others = plugInDialogDontDisplay / plugInDialogSilent
    if (playInfo != plugInDialogDisplay)
        {
        error = MakeNewDocument();
        if (error == kSPNoError)
            WriteScriptParams();
        }
    else
        {
        // Go ahead and display a dialog:

        error = DoUI();

        if (error == kSPNoError)
            {
            MakeNewDocument();
            WriteScriptParams();
            }
        else if (error == userCanceledErr)
            {
            error = kSPNoError;  // We've already reverted.
                                 // Don't need to return err.
            }
        }

    return error;

    }
```

Your `Execute` routine should use the `playInfo` flag to determine whether a user dialog box needs to be presented. To determine this, call `GetPlayInfo`. If `playInfo` is `plugInDialogDontDisplay`, or `plugInDialogSilent` then the `MakeNewDocument()` function is executed without a dialog box being displayed and the descriptor for this action is written via `WriteScriptParams()`. If `playInfo` is `plugInDialogDisplay`, then a user dialog box is displayed in `DoUI()`. If `DoUI()`returns successfully, `MakeNewDocument()` is executed and the descriptor for this action is written via the `WriteScriptParams()`.

`Initialize()` assigns an initial value for each variable.

```
static void Initialize (void)
    {
    gResolution = 72;
    gWidth = gResolution*4;
    gHeight = gResolution*3;
    gFill = enumWhite;
    gMode = classRGBColorMode;
    }
```

The default values are a four inch by three inch image with a resolution of 72 dots per inch.

`MakeNewDocument` performs the function of our new plug-in, creating a new document based on the user input.

```
static SPErr MakeNewDocument (void)
    {
    SPErr error = kSPNoError;

    PIActionDescriptor descriptor = NULL;
    error = sPSActionDescriptor->Make (&descriptor);

    if (error == kSPNoError)
        {
        error = sPSActionDescriptor->PutUnitFloat
            (
            descriptor,
            keyWidth,
            unitDistance,
            ConvertToUnitDistance(gWidth, gResolution)
            );
        }

    if (error == kSPNoError)
        {
        error = sPSActionDescriptor->PutUnitFloat
            (
            descriptor,
            keyHeight,
            unitDistance,
            ConvertToUnitDistance(gHeight, gResolution)
            );
        }

    if (error == kSPNoError)
        {
        error = sPSActionDescriptor->PutUnitFloat
            (
            descriptor,
            keyResolution,
            unitDistance,
            gResolution
            );
        }

    if (error == kSPNoError)
        {
        error = sPSActionDescriptor->PutEnumerated
            (
            descriptor,
            keyFill,
            typeFill,
            gFill
            );
        }

    if (error == kSPNoError)
        {
        error = sPSActionDescriptor->PutClass
            (
```

```
            descriptor,
            keyMode,
            gMode
            );
        }

    if (error == kSPNoError)
        {
        PIActionDescriptor playDescriptor = NULL;
        sPSActionDescriptor->Make(&playDescriptor);

        sPSActionDescriptor->PutObject
            (
            playDescriptor,
            keyNew,
            classDocument,
            descriptor
            );

        // I'm creating a new document, the event I
        // want should be called eventNewDocument or
        // something, right?  Nope.  Use eventMake.

        PIActionDescriptor resultDescriptor = NULL;

        error = sPSActionControl->Play
            (
            &resultDescriptor,
            eventMake,
            playDescriptor,
            plugInDialogDontDisplay
            );

        // Check for error here.  If an error occurred, there
        // will be a string in the descriptor, keyMessage, with
        // the error that occurred.

        if (resultDescriptor != NULL)
            sPSActionDescriptor->Free(resultDescriptor);

        if (playDescriptor != NULL)
            sPSActionDescriptor->Free(playDescriptor);

        if (descriptor != NULL)
            sPSActionDescriptor->Free(descriptor);

        } // error

    return error;

    } // end MakeNewDocument
```

The `MakeNewDocument` function starts by declaring an `error` variable and assigning it a `kSPNoError` value. If later on in this function, an error condition is encountered, this value can be changed to indicate an error condition. It can also be checked at various points to ensure that no errors have occurred.

The `MakeNewDocument` function next declares a variable `descriptor` of the type `PIActionDescriptor` and assigns it the value of `NULL`. It then

makes a function call `sPSActionDescriptor->Make (&descriptor)` to instantiate the new descriptor and place it in the location pointed to by the variable `descriptor`.

If no error was encountered, it then stuffs the new descriptor with the Width, Height, Resolution, Fill and Color Mode parameters by calling the `sPSActionDescriptor->PutUnit Float`, `sPSActionDescriptor->PutClass`, etc.

`sPSActionDescriptor->PutUnitFloat` is called with the parameters: `descriptor`, `keyWidth`, `unitDistance`, and `ConvertToUnitDistance(gWidth, gResolution)` which returns a double. `sPSActionDescriptor->PutUnitFloat` is defined in `PIActions.h` and is designed to put a floating point value of a specific type.

`PutUnitFloat` is called to store the height parameter, and again for the resolution parameter.

To put the fill type into the descriptor, use `sPSActionDescriptor->PutEnumerated` and to put the Color mode use `sPSActionDescriptor->PutClass`.

Next, we declare another descriptor variable `playDescriptor` of the type `PIActionDescriptor` and assign it the value of `NULL`. Then, as previously done with `descriptor`, the function `sPSActionDescriptor->Make (&playDescriptor)` is called to create the descriptor. The `playDescriptor` descriptor is not an object, but rather just a descriptor that holds another descriptor. `sPSActionDescriptor->PutObject()` creates the object with `playDescriptor`, `keyNew`, `classDocument`, and `descriptor` (the original descriptor created and stuffed with our MakeNew parameters earlier) as parameters.

```
PIActionDescriptor playDescriptor = NULL;
sPSActionDescriptor->Make(&playDescriptor);

sPSActionDescriptor->PutObject
    (
    playDescriptor,
    keyNew,
    classDocument,
    descriptor
    );

PIActionDescriptor resultDescriptor = NULL;

error = sPSActionControl->Play
    (
    &resultDescriptor,
    eventMake,
    playDescriptor,
    plugInDialogDontDisplay
    );
```

There is one more descriptor required before we play the `eventMake` command. This last descriptor (`resultDescriptor`) is declared and will receive the results:

```
PIActionDescriptor resultDescriptor = NULL;
```

Finally, with the container descriptor (`playDescriptor` -- that also contains our original stuffed descriptor variable) and the `resultDescriptor` ready, we can now play the `eventMake` command to actually make it happen.

Note: you don't have to `Make()` this descriptor because it will be handed to you from the host. You must free it if you are given one, however.

```
error = sPSActionControl->Play
            (
            &resultDescriptor,
            eventMake,
            playDescriptor,
            plugInDialogDontDisplay
            );
```

Calling `Play()` actually performs the event, as long as the descriptor contains the correct information for the event. If it doesn't play, or returns an error, try checking to make sure that you are using the correct types and keys for the parameters for the event.

After playing the eventMake action, it returns a descriptor in `resultDescriptor`, the original descriptor is still in `playDescriptor`, and the original descriptor variable `descriptor` still contains parameters of our document.

You must free all descriptors you create or the host hands to you. After checking to make sure that they are not `NULL`, the sample plug-in calls `sPSActionDescriptor->Free(descriptor)` to free each descriptor.

The main body of the plug-in code is finished. All it does is read an incoming descriptor, determine whether or not to display a dialog box, perform `MakeNewDocument()` based on the parameters, then hands a new descriptor with those parameters back to the host. `MakeNewDocument()` creates three new descriptors, stuffs one of them with our new document parameters, stuffs that into a second descriptor, creates a result descriptor, calls `eventMake` (which actually creates the new document using the Photoshop Actions Engine) and then frees up the descriptors.

Note that the actual work of reading and writing descriptors is handled in the routines in the `MakeNewScripting.cpp` source file, while the dialog resources are defined in the `MakeNew.r` file and the dialog handling is performed in the `MakeNewUI.cpp` source file. While not required, it is suggested that you follow a similar modular source partitioning scheme.

Now that we have our basic function code correctly set up, we need to address the scripting source code.

Step 9 - Revise `MakeNewScripting.cpp` source file to eliminate unnecessary TriggerFilters prototypes and functions and to add the specific descriptor information we need for the make a new document: Width, Height, Resolution, Fill, ColorMode. This is where your plug-in will interact with the Photoshop Actions descriptor data structures.

`MakeNewScripting.cpp` performs the actual reading and writing of descriptors. It contains the usual includes, `PIDefines.h` and `MakeNew-PPC.ch` or `MakeNew.h` and the `ReadScriptParams()` and `WriteScriptParams()` functions.

```
SPErr ReadScriptParams ()
{
    SPErr error = kSPNoError;

    if (sPSActionDescriptor.IsValid())
        { // Make sure we have a valid suite before trying this.
        PIActionDescriptor descriptor = NULL;
        PIDialogPlayOptions playInfo = plugInDialogDontDisplay;

        sPSActionDescriptor.GetPlayInfo(&descriptor, &playInfo);

        // If we got a valid descriptor, grab our key out of it:
        if (descriptor != NULL)
            {
            error = sPSActionDescriptor->GetInteger
                (
                descriptor,
                keyMyWidth,
                &gWidth
                );

            error = sPSActionDescriptor->GetInteger
                (
                descriptor,
                keyMyHeight,
                &gHeight
                );

            error = sPSActionDescriptor->GetInteger
                (
                descriptor,
                keyMyResolution,
                &gResolution
                );

            // We don't care about type, so declare a variable
            // that we'll ignore:
            DescriptorEnumTypeID enumType = enumNull;

            error = sPSActionDescriptor->GetEnumerated
                (
                descriptor,
                keyMyFill,
                &enumType,
                &gFill
                );

            error = sPSActionDescriptor->GetClass
                (
```

```
                descriptor,
                keyMyMode,
                &gMode
                );

            // Since we were handed this descriptor, it's our job to
            // free it, but we'll free it on SetReturnInfo, so we're okay.

            } // descriptor == NULL

        }

    return error;

} // end ReadScriptParams
```

`ReadScriptParams()` reads a new descriptor for our Make New Document plug-in. It tests to make sure the `sPSActionDescriptor` suite is valid before we start executing. **Note:** The class: `sPSActionDescriptor` and its member: `IsValid()` are defined in `PIUSuitePointer.h`.

If the suite is valid, we declare a `descriptor` of the type `PIActionDescriptor` and assign it a `NULL` value. Then we declare a `playinfo` of the type `PIDialogPlayOptions` and assign it the value of `plugInDialogDontDisplay`. It then calls `sPSActionDescriptor.GetPlayInfo(&descriptor, &playinfo)` to acquire the descriptor information and dialog display information and places this information in the variables `descriptor` and `playinfo`.

We next check to see if there is any information in the descriptor. It should have some, so we proceed to read the various keys for our Make Document function: Width, Height, Resolution, Fill Mode, and Image Mode. The descriptor is dynamic, sizing to fit the data it contains. You have to anticipate what data is coming and call the correct functions to acquire your data.

For the `MakeNew` plug-in, this means anticipating what types of keys we will be acquiring from the incoming descriptor. We know that we want Width, Height, Resolution, Fill Mode, and Image Mode.

We have defined our key values in our `MakeNewTerminology.h` file as:

```
#define keyMyWidth       keyWidth
#define keyMyHeight      keyHeight
#define keyMyResolution  keyResolution
#define keyMyFill        keyFill
#define typeMyFill       typeFill
#define keyMyMode        keyMode      // RGB, CMYK, etc.
#define typeClassMyMode  typeClassMode
```

Since our values are nicely mapped into Photoshop's existing keys for Width, Height, etc. we are in good shape and we can quickly acquire the first three using `GetUnitFloat()`. The fourth value, Fill, is an enumerated value and enumerated values require an enumType variable in addition

to the key and value. Therefore, we must declare an enumType variable to hold the incoming type information, even though we don't care about it. The declaration and assignment statement:

```
DescriptorEnumTypeID enumType = enumNull;
```

creates the variable with a default value and `sPSActionDescriptor->GetEnumerated`() acquires the enumerated Fill value.

The final information we need from the incoming descriptor is the color mode. This information is a Photoshop class and thus the `sPSActionDescriptor->GetClass()` function is used to read this value.

At this point, we have acquired a new descriptor and read all of the values needed for `MakeNewDocument()` in `MakeNew.cpp` to execute successfully.

`WriteScriptParams()` is the mirror image of `ReadScriptParams()`, with `Put` functions in place of `Get` functions.

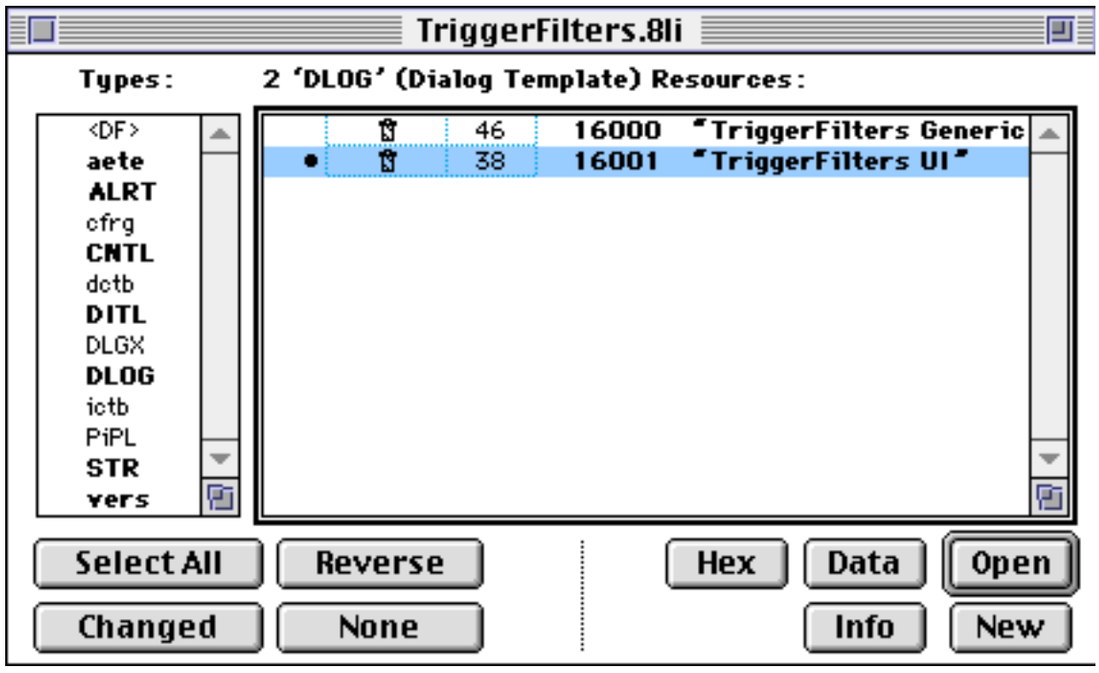
Now, we need to handle the user interface and the resources associated with the dialog boxes. This is done by revising the resource file and `MakeNewUI.cpp` file.

Step 10 - Revise the `MakeNew.r` file to reflect the desired user interface.

The `MakeNew.r` file is the resource file that holds the descriptions of three essential resources for our plug-in: the PiPL resource, the `'aete'` dictionary resource, and the dialog resource. We use two tools to create a binary dialog with a visual editor. Then we'll translate our revision into text and copy it into the file `MakeNew.r` resource file.

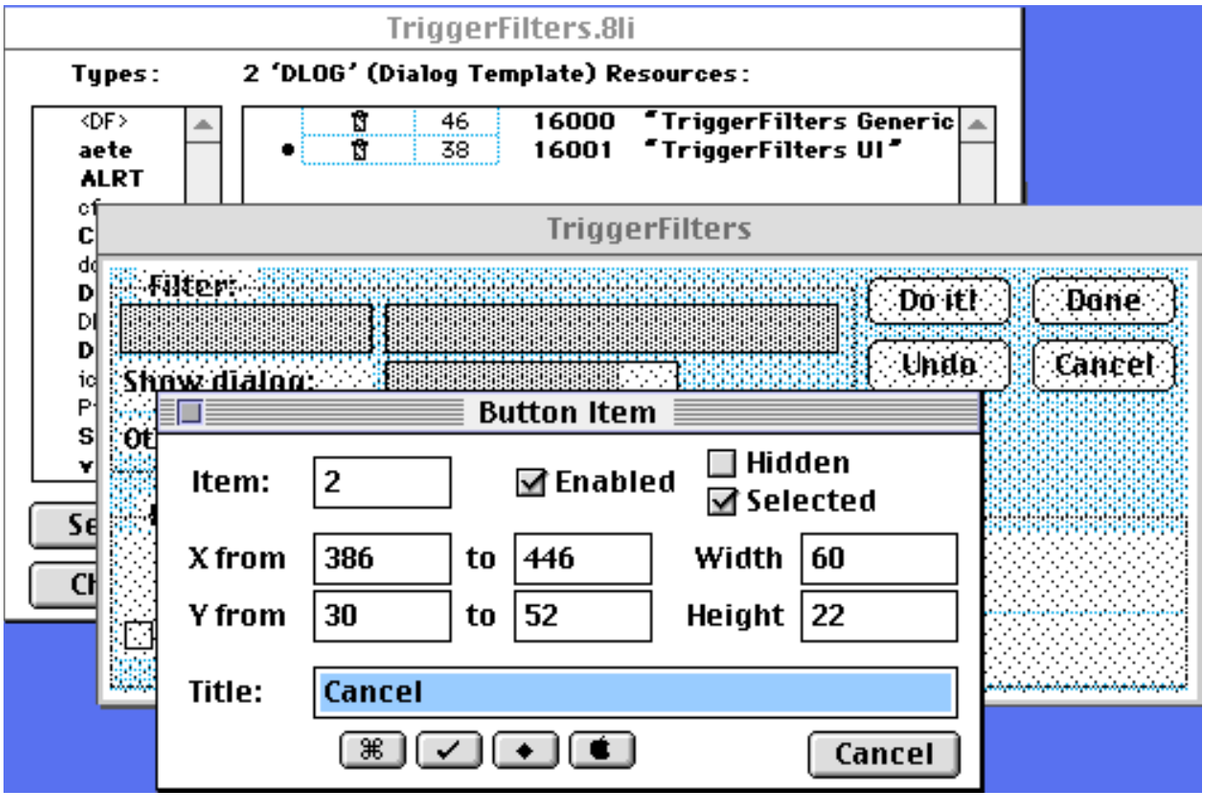
This process is simplified by the use of Resorcerer® from Mathemaesthetics, Inc. Resorcerer is an excellent Macintosh tool for editing file resources, especially dialog boxes. For Windows developers, this capability is already built into Microsoft Development Studio.

In Resorcerer, you'll see:



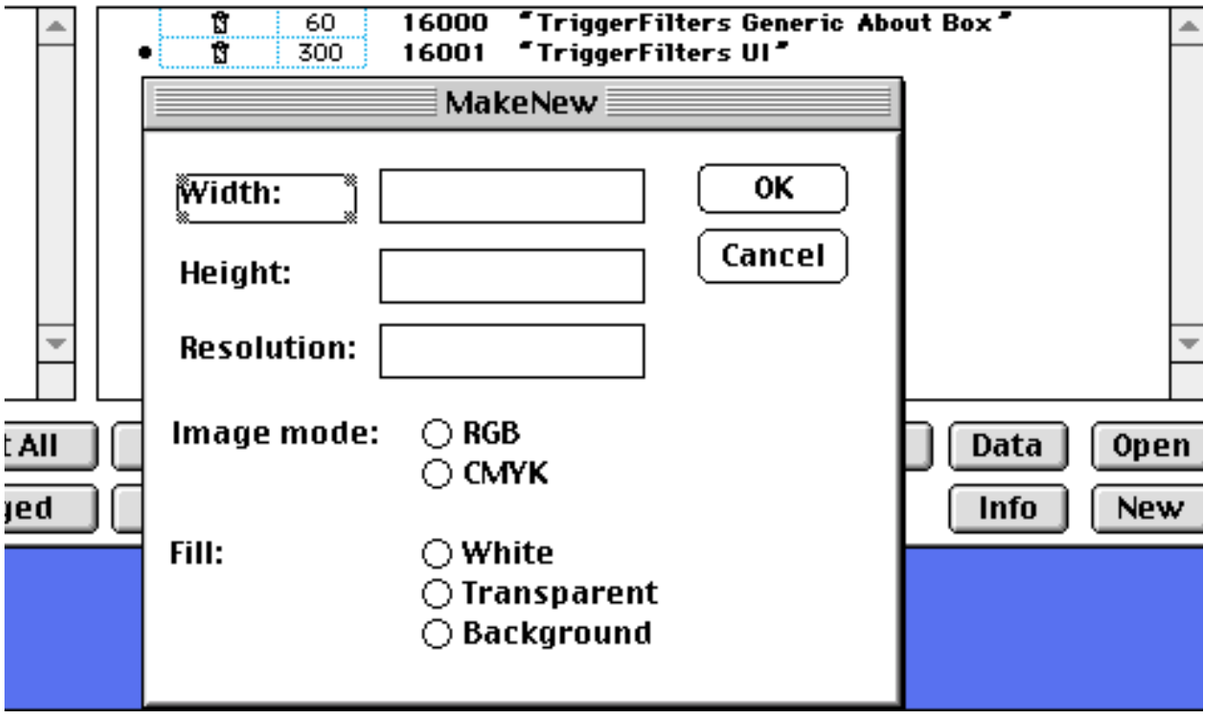
Make new `DLOG` and `DITL` resources that define the dialog box and internal dialog items. Then arrange the dialog box resources to suit your needs in the visual editor.

With Resorcerer, you can easily edit the dialog box and rearrange the various elements: edit text boxes, static text boxes, input text boxes, radio buttons, etc.



When you are done with the dialog window, save your revised file as `MakeNew-derez.rsrc` or with the name of your new plug-in.

Our newly revised dialog box will look like:



In our MakeNew plug-in, we have created Width, Height, Resolution, Image Mode and Fill options. Width, Height and Resolution each require a Static Text box and a Edit Text box. Image Mode requires two radio buttons for RGB and CMYK options. Fill requires Static Text and two radio button background options: White and Transparent. We are limiting our options for simplicity sake.

Resorcerer lets you test out your dialog boxes and rearrange them until they fit they way you want. When you are finished, save your binary file as `MakeNew-derez.rsrc`.

Step 11 - Revise the `MakeNew-derez.rsrc` file to include the correct PiPL, `'aete'`, and dialog box resources.

Once you are satisfied with the look of your dialog box using Resorcerer, run the CodeWarrior ToolServer MPW Derez script to turn the binary resource code into a resource text file. This script is provided for you in the `MakeNew.derez` file. When you run this script under CodeWarrior ToolServer, it changes any binary resources in the input file (in this case: `MakeNew-derez.rsrc`) into readable text that can be easily edited. We name our MakeNew output resource text file: `MakeNew-derez.r`.

NOTE: `MakeNew.derez` is a script file that runs under CodeWarrior ToolServer. See the CodeWarrior documentation for more information on this tool.

In Resorcerer, we can manipulate the dialog boxes and other items in their visual representation. Using Derez, we can deal with them as text items.

The MakeNew.derez.r file holds our new resource text file that describes the dialog box resources. It contains the Macintosh DLOG and DITL dialog box information. Note that the DLOG resource describes the entire dialog box information (size, visible, has go away box, etc.) while the DITL resource describes the individual elements within the dialog box (each button, text field, etc.).

```
resource 'DLOG' (16000, "", purgeable) {
    {20, 0, 214, 390},
    movableDBoxProc,
    visible,
    noGoAway,
    0x0,
    16000,
    ""
    /****** Extra bytes follow... ******/
    /* $"A80A"                                                 /* ®. */
};

resource 'DLOG' (16001, "", purgeable) {
    {87, 27, 307, 319},
    movableDBoxProc,
    visible,
    noGoAway,
    0x0,
    16001,
    "MakeNew"
    /****** Extra bytes follow... ******/
    /* $"680A"                                                 /* h. */
};

resource 'DITL' (16000, "", purgeable) {
    {   /* array DITLarray: 3 elements */
        /* [1] */
        {-80, 0, -60, 60},
        Button {
            enabled,
            "Hidden"
        },
        /* [2] */
        {0, 0, 194, 390},
        UserItem {
            enabled
        },
        /* [3] */
        {5, 5, 190, 360},
        StaticText {
            disabled,
            "^0\n^1\n^2\n"
        }
    }
};

resource 'DITL' (16001, "", purgeable) {
    {   /* array DITLarray: 15 elements */
        /* [1] */
        {8, 216, 28, 276},
        Button {
            enabled,
            "OK"
        },
```

```
/* [2] */
{34, 216, 56, 276},
Button {
    enabled,
    "Cancel"
},
/* [3] */
{12, 8, 32, 80},
StaticText {
    disabled,
    "Width:"
},
/* [4] */
{13, 92, 29, 192},
EditText {
    enabled,
    ""
},
/* [5] */
{45, 92, 61, 192},
EditText {
    enabled,
    ""
},
/* [6] */
{44, 8, 64, 80},
StaticText {
    disabled,
    "Height:"
},
/* [7] */
{74, 8, 94, 88},
StaticText {
    disabled,
    "Resolution:"
},
/* [8] */
{75, 92, 91, 192},
EditText {
    enabled,
    ""
},
/* [9] */
{108, 4, 140, 96},
StaticText {
    disabled,
    "Image mode:"
},
/* [10] */
{108, 104, 124, 264},
RadioButton {
    enabled,
    "RGB"
},
/* [11] */
{124, 104, 140, 264},
RadioButton {
    enabled,
    "CMYK"
},
/* [12] */
{156, 4, 188, 80},
StaticText {
```

```
            disabled,
            "Fill:"
        },
        /* [13] */
        {156, 104, 172, 264},
        RadioButton {
            enabled,
            "White"
        },
        /* [14] */
        {172, 104, 188, 264},
        RadioButton {
            enabled,
            "Transparent"
        },
        /* [15] */
        {188, 104, 204, 264},
        RadioButton {
            enabled,
            "Background"
        }
    }
};
```

At this point, we will copy the `DLOG` and `DITL` resources from this file and put them in our `MakeNew.r` file. This file is just a renamed `TriggerFilters.r` file, so we need to revise all of the items for our MakeNew resources. Before we revise the `DLOG` and `DITL` resources, we'll first revise the PiPL and `'aete'` resources.

A Brief Note on ADM and Dialog Element Management

Even though we are using ADM to handle our dialog resources in our plug-in, we still need to create (or revise) the dialog box resources themselves. While we must create the actual dialog resources on each platform, we can greatly simplify our programming overhead by using ADM to handle how we interact with the dialog elements. ADM Suites and functionality are documented in *ADM.pdf*.

Step 12 - Revise the `MakeNew.r` resource file to update the resources for the MakeNew plug-in. `MakeNew.r` starts with the following definitions:

```
// The About box and resources are created in DialogUtilities.r.
// You can easily override them, if you like.

#define plugInName            "MakeNew"
#define plugInCopyrightYear   "1998"
#define plugInDescription \
    "An example Actions Module to make a new document in Adobe Photoshop®."

// Dictionary (aete) resources:

#define vendorName            "AdobeSDK"
#define plugInAETEComment     "makenew example actions plug-in"

#define plugInSuiteID         'sdKF'
#define plugInClassID         plugInSuiteID
#define plugInEventID         'makN'

//  Set up included files for Macintosh and Windows.
```

```
#include "PIDefines.h"

#ifdef __PIMac__
    #include "Types.r"
    #include "SysTypes.r"
    #include "PIGeneral.r"
    #include "PIUtilities.r"
    #include "DialogUtilities.r"
#elif defined(__PIWin__)
    #include "PIGeneral.h"
    #include "PIUtilities.r"
    #include "WinDialogUtils.r"
#endif

#include "PIActions.h"
#include "MakeNewTerminology.h"
```

These resource code and definitions get compiled and become the resources for our `MakeNew` plug-in.

The first group of defines name our plug-in name “MakeNew”, sets the copyright year, and the description. The dictionary `'aete'` resources are set as `AdobeSDK`, etc.

Note: You must replace these SDK values. SDK_ is reserved by Adobe.

Finally, we will include a number of `.r` files (`Types.r`, `PIGeneral.r`, etc.). These are the templates for our resources.

Resource files (`FILENAME.r`) require a template as well as the resource file. The `.r` file contains definitions that cannot compile without the template. Templates set the format required for the resource files. This is why the include files are necessary.

For example, the base template file for resources of type `'PiPL'` is `PiPL.r`. This file is found in the SDK in *SDK/Macintosh/SampleCode/Common/Rez-files/Photoshop* folder and is included in the `PIGeneral.r` file. It defines what a PiPL format is and what parameters are required. The `MakeNew.r` file will contain a PiPL (as well as other resources) that adheres to the template defined in `PiPL.r`.

In the `MakeNew.r` file, we have four important resources: 1) the PiPL, 2) an `'aete'` dictionary resource, 3) the `DLOG` and `DITL` dialog resources, and 4) any custom strings we may have.

In the `MakeNew.r` file, we will rewrite any items that refer to the TriggerFilters plug-in. This means that we will revise the PiPL information, the `'aete'` definition, the dialog resources (`DLOG` and `DITL`) and some of our string information.

1) Edit the PiPL

We start with the PiPL. This is the static information that tells any host information about your plug-in. PiPLs contain the properties of your plug-in.

The first entry in the PiPL is the version. The version field contains the version of the PiPL template. The current version as of Photoshop 5.0 is 0. The count field holds the number of properties contained in the PiPL. The properties field is a variable length array containing the actual PiPL data.

Each property field contains at minimum five items: a vendorID code that identifies the host of the plug-in; a propertyKey that specifies the type of plug-in; a propertyID that is always zero except in one rare exception and can be considered reserved; a propertyLength that contains the length of the propertyData field; and propertyData, a variable length field that contains the plug-in specific data.

PiPLs are more completely documented in the *Cross-application Plug-In Resource Guide* document contained in the Photoshop 5.0 SDK. The `PiPL.r` file also contains a complete template for all Adobe PiPLs.

```
resource 'PiPL' (ResourceID, plugInName " PiPL", purgeable)
    {
        {
        Kind { Actions },
        Name { plugInName "..." },
        Category { "" },
        Version { (latestActionsPlugInVersion << 16)
                 | latestActionsPlugInSubVersion },

        #ifdef __PIMac__
            CodePowerPC { 0, 0, "" },
        #elif defined(__PIWin__)
            CodeWin32X86 { "ENTRYPOINT" },
        #endif

        HasTerminology
            {
            plugInClassID,
            plugInEventID,
            ResourceID,
            vendorName " " plugInName// Unique string.
            },

        // If you want this on all the time,
        // remove the EnableInfo property (such
        // as for help menu entries.)  To have this
        // on according to document
        // open, close, and mode guidelines,
        // provide a minimal EnableInfo, such
        // as the one here:
        // I'm going to comment EnableInfo out so that this plug-in
        // is available all the time
        // (absence of EnableInfo = always available).
        // EnableInfo { "true" },
        }
    };
```

Starting from the top, note that we can use “Actions” as our plug-in kind because it is defined in the `PiPL.r` template file as follows:

```
#define PIPiPLTypes                 \
    /* Photoshop plug-in types: */\
    General = '8BPI',               \
    Filter = '8BFM',                \
    Parser = '8BYM',                \
    ImageFormat='8BIF',             \
    Extension = '8BXM',             \
    Acquire = '8BAM',               \
    Export = '8BEM',                \
    Selection = '8BSM',             \
    Picker = '8BCM',                \
    Actions = '8LIZ',               \ //here is our plug-in type definition!
```

Since Action is defined as `'8LIZ'`, we can use either Action or `'8LIZ'` as our plug-in kind.

For Name, we use

`'plugInName "..."'`.

This will automatically be filled in for you because it is defined at the top of the file, and this is what will be shown in the Automate menu (MakeNew ...).

There is no Category value, so it is left blank. If you wanted to put your Plug-in in the Help menu and not the Automate menu, you would put the value, `PSHelpMenu` into the Category element. With no Category value, your plug-in will appear in the Automate menu. If you do not want your automation plug-in to appear in any menu (which would be the case if you wanted to call your plug-in from another plug-in without letting the user call it directly), put the value `PSHidden` in the Category element. Be careful, however, since if you hide it, your user will have no way of calling it.

The `HasTerminology` structure tells Photoshop that you have an `'aete'` dictionary (to be defined later in this file), and it includes ClassID, EventID, ResourceID and a unique string. The unique string can be your vendor name plus your plug-in name, but we recommend UUIDs. These elements are defined at the top of this file.

At the end of the PiPL, there is an EnableInfo { } element. This element determines when your plug-in is made available to the user. (When your plug-in is not available, its menu entry will be greyed out.). In the absence of an `EnableInfo` data structure, the plug-in will be available all the time. If `EnableInfo` is set to “true”, it will be available all the time if there is a document open. You can use this setting to determine when your plug-in is enabled. For example, if `EnableInfo` is set to `In(PShop_ImageMode(RGBMode, CMYKMode))` then the plug-in would be enabled only when the image mode is RGB or CMYK. See the *Cross-app Plug-in Resource Guide* for more information.

Since our MakeNew plug-in needs to be available even when there is no document open, we will eliminate the `EnableInfo` property entirely, making the plug-in always available.

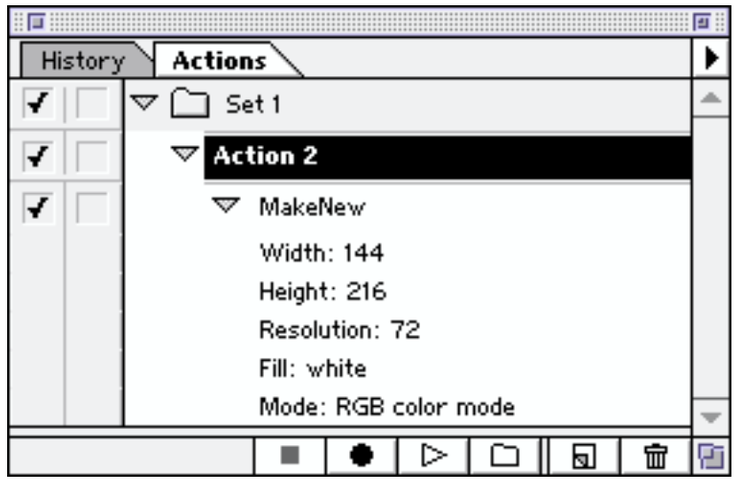
The last thing in this PiPL is the entry point of the plug-in. You leave this untouched, since it is generated automatically by the compiler.

2) Editing dictionary resources

The `'aete'` is a dictionary resource required for AppleScript and the Photoshop scripting system is based on AppleScript. The `'aete'` resource is required. It is a dictionary resource that defines various elements of your plug-in parameters and maps your text names to the actual values they represent. You can copy the TriggerFilters `'aete'` resource and fill in the sections that apply to your plug-in. For an automation plug-in, you need to fill in the top part and the parameter values.

In the case of MakeNew, we insert our unique variables (plugInName, plugInClassID, plugInEventID, etc.) that are defined at the top of this file. The `'aete'` requires that we list the parameters and their key IDs, so we put in “width”, keyMyWidth, “height”, keyMyHeight, etc. This maps our keys to “human readable text.” For example, our key “keyMyWidth” is mapped to the human readable text: “width”, and so on.

NOTE: The text values that you enter in the `'aete'` will be the text that is shown in the Actions Palette when your automation plug-in runs. Since we mapped the text “width” to our value keyMyWidth, if we capture our plug-in execution with the Record Actions command, the value shown in the Action Palette will be “width.”



Finally, the `'aete'` requires that we put our enumerated values separately from the other parameters. So we put “white” and “transparent” at that location. That completes the MakeNew `'aete'` resource.

```
resource 'aete' (ResourceID, plugInName " dictionary", purgeable)
    {
    1, 0, english, roman,/* aete version and language specifiers */
        {
        vendorName,                   /* vendor suite name */
        "Adobe example plug-ins",     /* optional description */
```

```
plugInSuiteID,                /* suite ID */
1,                            /* suite code, must be 1 */
1,                            /* suite level, must be 1 */
    {                         /* structure for automation */
    plugInName,               /* name */
    plugInAETEComment,        /* optional description */
    plugInClassID,       /* class ID, must be unique or Suite ID */
    plugInEventID,            /* event ID, must be unique */

    NO_REPLY,                 /* never a reply */
    IMAGE_DIRECT_PARAMETER,/* direct parameter, used by Photoshop */
        {                     // filter or selection class here:
        // name:
        "width",
        // key ID:
        keyMyWidth,
        // type ID:
        typeInteger,
        // optional description:
        "",
        // flags:
        flagsSingleParameter,

        "height",
        keyMyHeight,
        typeInteger,
        "",
        flagsSingleParameter,

        "resolution",
        keyMyResolution,
        typeInteger,
        "",
        flagsSingleParameter,

        "mode",
        keyMyMode,
        typeClassMyMode,
        "",
        flagsSingleParameter,

        "fill",
        keyMyFill,
        typeMyFill,
        "", // optional comment
        flagsEnumeratedParameter

        }
    },
    {}, /* non-filter/automation plug-in class here */
    {}, /* comparison ops (not supported) */
    { // Enumerations go here:
        typeFill,
        {
            "white",
            enumWhite,
            "",

            "transparent",
            enumTransparent,
            ""
        }
    }   /* end of any enumerations */
```

```
        }
    };
```

3) Edit the Dialog resources

We are now ready to revise the `DLOG` and `DITL` resources that we copied into our `MakeNew.r` file from the `MakeNew.derez.r` file.

`DLOG` is the general information about the dialog box, its size, when it is visible, whether has a title bar that is movable, whether it can be closed (it has a go away box), where in the screen it will be displayed, etc. `DITL` contains the descriptions of the individual items in the dialog window.

`DLOG` - In our modified `DLOG` resource, we just need to change the plug-inName. For example, the Trigger Filters `DLOG` resource looks like:

```
resource 'DLOG' (16001, "", purgeable) {
    {87, 27, 307, 319},
    movableDBoxProc,
    visible,
    noGoAway,
    0x0,
    16001,
    "MakeNew"
    /****** Extra bytes follow... ******/
    /* $"680A"                                              /* h. */
};
```

We'll change it to look like:

```
resource 'DLOG' (uiID, plugInName " UI", purgeable) {
    {87, 27, 307, 319},
    movableDBoxProc,
    visible,
    noGoAway,
    0x0,
    uiID,
    plugInName,
    centerParentWindowScreen
};
```

Here we are changing the first line from static text to variables that we can define elsewhere. This simplifies the process. We could have also specified the correct text as:

```
resource 'DLOG' (16001, “MakeNewUI", purgeable) {
```

but this hard coding requires that we go and find each instance of the hard coded text when we want to make a change. By using variables, we can just change the definitions once and the correct information will be inserted everywhere it is required.

NOTE: The resource uiID is set at 16001 because Adobe Developer Relations chose to use this number as a unique resource ID. There is no significance to this number. It is defined in `PIUBasic.h`.

`DITL` - In the `DITL` resource, we need to change all of the elements that we are going to use in your dialog box.

```
resource 'DITL' (uiID, plugInName " UI", purgeable) {
    {   /* array DITLarray: 14 elements */
        /* [1] */
        {8, 216, 28, 276},
        Button {
            enabled,
            "OK"
        },
        /* [2] */
        {34, 216, 56, 276},
        Button {
            enabled,
            "Cancel"
        },
        /* [3] */
        {12, 8, 32, 80},
        StaticText {
            disabled,
            "Width:"
        },
        /* [4] */
        {13, 92, 29, 192},
        EditText {
            enabled,
            ""
        },
        /* [5] */
        {44, 8, 64, 80},
        StaticText {
            disabled,
            "Height:"
        },
        /* [6] */
        {45, 92, 61, 192},
        EditText {
            enabled,
            ""
        },
        /* [7] */
        {74, 8, 94, 88},
        StaticText {
            disabled,
            "Resolution:"
        },
        /* [8] */
        {75, 92, 91, 192},
        EditText {
            enabled,
            ""
        },
        /* [9] */
        {108, 4, 140, 96},
        StaticText {
            disabled,
            "Mode:"
        },
        /* [10] */
        {108, 104, 124, 264},
        RadioButton {
```

```
            enabled,
            "RGB"
        },
        /* [11] */
        {124, 104, 140, 264},
        RadioButton {
            enabled,
            "CMYK"
        },
        /* [12] */
        {156, 4, 188, 80},
        StaticText {
            disabled,
            "Fill:"
        },
        /* [13] */
        {156, 104, 172, 264},
        RadioButton {
            enabled,
            "White"
        },
        /* [14] */
        {172, 104, 188, 264},
        RadioButton {
            enabled,
            "Transparent"
        }
    }
};
```

Note the change in the first line:

```
resource 'DITL' (16001, "TriggerFilters UI", purgeable) {
```

becomes:

```
resource 'DITL' (uiID, plugInName " UI", purgeable) {
```

Of course, the individual data elements for each button are unique to the MakeNew and TriggerFilter plug-in resources. For the MakeNew plug-in, there are individual dialog elements for the OK button, the Cancel button, the static text buttons, edit texts for Width, Height, Resolution, and the radio buttons for Fill and Color Mode.

For more information on dialog resources, refer to Macintosh and Windows programming reference guides.

4) Edit the String Resources

You can put text in the dialog boxes as literal text. However, every time you changed your text strings, the dialog resources would have to be adjusted manually. By using string resources, you can change your text without affecting other elements of the program. (This is very useful for localization of your plug-in for international markets.)

The Cancel button is also used when the user holds option or alt, as a “Reset” button. In the MakeNew plug-in we use two strings for “Reset” and Cancel. Using strings is not just a great time saver for adjusting your

plug-in for international use, but it also lets you keep memory allocation of literally defined strings down to a minimum. It is also helpful for easily converting this information from Macintosh to Windows platforms.

```
resource StringResource (kResetStringID, purgeable)
    {
    "Reset"
    };

resource StringResource (kCancelStringID, purgeable)
    {
    "Cancel"
    };
```

These strings are used to rename the Cancel button. It is a Photoshop historical feature to let the user reset to default values by placing the cursor over the Cancel button while holding down the option or alt key. These strings will be used to implement this function.

This completes the revision of the MakeNew.r resource file. It is a good practice to compile this file now and find and eliminate any compile time errors. If we have none, then we know we have a good resource file for our project.

We now have revised our header files, the main `MakeNew.cpp` source file, the `MakeNewScripting.cpp` source file and the `MakeNew.r` resource file. Now all we need do to complete our plug-in is to revise our `MakeNewUI.cpp` source file.

Step 13 - Revise the `MakeNewUI.cpp` source file. This file will handle all of the UI functions, using calls to ADM suites to manage how to respond to where the mouse is positioned and what buttons and keys the user presses. ADM Suites and their functionality are documented in *ADM.pdf* contained in the Photoshop 5.0 SDK. One advantage of ADM is we write the user interface handler once and can use it on both Mac and PC platforms.

Our current `MakeNewUI.cpp` source file is just a copy of the `TriggerFiltersUI.cpp` file, so we have to go in and eliminate the references to TriggerFilters resources and replace them with the references to the MakeNew resources as defined in `MakeNew.r`.

`MakeNewUI.cpp` starts with the usual defines files that we need to revise to use the `MakeNew.h` file, and then lists the definitions of our dialog items. These are an enumerated list starting with the “OK” button as item 1, and the Cancel button is item 2, so we just have to list the constants associated with each dialog item. These items can be found in `MakeNew.r` file.

Note: ADM requires and expects that items numbered 1 and 2 will be the “Okay” button and Cancel button in that order.

```
enum
    {
    kDNoUI = -1,// Error.
    kDOk_button = 1,// Must be one.
    kDCancel_button,// Must be two.
    kDWidth_staticText,
    kDWidth_editText,
    kDHeight_staticText,
    kDHeight_editText,
    kDResolution_staticText,
    kDResolution_editText,
    kDMode_staticText,
    kDMode_button_RGB,
    kDMode_button_CMYK,
    kDFill_staticText,
    kDFill_button_white,
    kDFill_button_transparent
    };
```

The “D” designation is just an Adobe Developer Relations convention to indicate that these constants are dialog box related.

Next are the constants associated with MakeNew.

```
const int kHeightMin = 1;
const int kHeightMax = 30000;

const int kWidthMin = kHeightMin;
const int kWidthMax = kHeightMax;

const int kResolutionMin = 1;
const int kResolutionMax = 9999;
```

In this case, we are defining a maximum and minimum value for height and width, and are further defining these values for Width to be the same as those for Height. Finally, we are setting the Resolution to be a value from 1 to 9999 (Whew! that would be a big file!). Because we are also going to be concerned with the state of the Cancel button (see code explanation later on in this chapter), we will declare a boolean variable: `gCancelButtonIsReset` and set it’s value to `false`. This will be used later to determine the state of the Cancel button.

Next are the prototypes for our “handle the user input” routines:

```
static ASErr ASAPI DoUIInit(ADMDialogRef dialog);

static SPErr InitCancelButton(ADMDialogRef dialog);

static void SaveParameters(void);
static void RestoreParameters(void);
static void SetTextToCancel(ADMItemRef item);
static void SetTextToReset(ADMItemRef item);

static void ASAPI DoCancel(ADMItemRef item, ADMNotifierRef notifier);
static ASBoolean ASAPI DoReset(ADMItemRef item, ADMTrackerRef tracker);

static void ASAPI DoWidthEditText(ADMItemRef item, ADMNotifierRef notifier);
static void ASAPI DoHeightEditText(ADMItemRef item, ADMNotifierRef notifier);
```

```
static void ASAPI DoResolutionEditText(ADMItemRef item,
                                                ADMNotifierRef notifier);
```

These include calls to ADM functions and start with an initialize function, an initialize the cancel button function, save and restore parameters functions, and set text to Cancel or Reset functions.

Next comes the prototypes for the actual Cancel and Reset functions, as well as the input text for our dialog boxes for Width, Height, and Resolution options.

Our global values are defined and then the various UI routines are listed, starting with `DoUI` shown below. `DoUI` sets up ADM to handle our user interface interaction.

```
SPErr DoUI ()
    {
    SPErr error = noErr;

    int item = -1;// Error value.

    SaveParameters();// Save our parameters, just in case.

    if (sPSUIHooks.IsValid() && sADMDialog.IsValid())
        {
        item = sADMDialog->Modal
            (
            sPSUIHooks.GetPlugInRef(),
            "MakeNew",
            uiID,
            kADMModalDialogStyle,
            DoUIInit,
            NULL /* No user data */
            );
        }

    if (item != kDOk_button)
        {
        error = userCanceledErr;
        RestoreParameters();
        }

    return error;

    }
```

The first thing performed in the `DoUI` routine is the creation of an integer variable, `item`, and setting it to the value of `-1` (an error value). This is because this routine returns an integer value and we want to make sure that if there is a problem in this routine, we return an error value.

We next check to make sure that the Photoshop User Interface Hooks and ADM Dialog suites are valid:

```
    if (sPSUIHooks.IsValid() && sADMDialog.IsValid())
```

If they are valid, then `item` is assigned the value returned by the `sADMDialog->Modal` function call, which is the number of the dialog item that was used to dismiss the dialog. Usually `kDOk_button` or `kDCancel_button`.

`sADMDialog` is a smart suite pointer for the suite that runs the dialog interaction. `sPSUIHooks` is another pointer to the suite that contains some utility functions including `sPSUIHooks.GetPlugInRef()` which we use to get the unique Plug-in Reference ID that is given to the plug-in by the host when the plug-in is first started. Several ADM suites and functions require the plug-in reference of the current plug-in. ADM uses that information to track who owns what dialog.

In `sADMDialog->Modal()` we pass several parameters as defined in the `ADMDialog.h` file. They set up ADM to handle the dialog functions with the following information:

1) The `sPSUIHooks.GetPlugInRef()` function provides the unique plug-in reference.

2) A name of the new dialog is provided. In this case, we use “MakeNew Document”. This is the text that will appear in the title bar of this new dialog box.

3) A dialog ID (`uiID`) is provided. It is the same ID as used in the `MakeNew.r` file. This tells ADM what resource to use for the dialog. The cross-platform ADM routines will draw the user interface using the platform specific dialog resources defined earlier. In our plug-in, `uiID` is defined as 16001. (See `PIUtilities.r` for those define statements.)

4) A constant that specifies the style of dialog. In this case (and in almost all Photoshop plug-ins), `kADMModalDialogStyle` is used. This dialog style supports a window that can be moved around the screen, and that must be dismissed before continuing.

5) The name of the routine that initializes the dialog elements using ADM routines. In this case, it is our `DoUIInit()` routine.

6) Any extra data that may be required. This information can be a pointer, strings, structures, or whatever you may want to associate with your dialog boxes. In this case, there is no user data, so we pass a `NULL` value.

This routine, `DoUI()`, runs the dialog, does the interaction with the dialog boxes, and when the user hits the Ok or Cancel button, it returns with the value of the button used to end the routine in the variable `item`. This is shown below:

```
if (item != kDOk_button)
    {
    error = userCanceledErr;
    RestoreParameters();
    }
```

If the button used to dismissed the dialog is not the Ok button, then it must have been the Cancel button, so this routine returns `userCanceledErr` and restores the old parameters. Otherwise, if the routine went okay, then there is no error returned.

The first function that is executed when `DoUI()` is run is `DoUIInit`:

```
static ASErr ASAPI DoUIInit(ADMDialogRef dialog)
{
    ASErr error = kSPNoError;

    ADMItemRef item;

    // Set up list and display default item:
    InitCancelButton(dialog);

    // Set up Width edit text:
    item = sADMDialog->GetItem(dialog, kDWidth_editText);
    sADMItem->SetUnits(item, kADMNoUnits);
    sADMItem->SetIntValue(item, (int)gWidth);
    sADMItem->SetMinIntValue(item, kWidthMin);
    sADMItem->SetMaxIntValue(item, kWidthMax);
    sADMItem->SetNotifyProc(item, DoWidthEditText);

    // Set up Height edit text:
    item = sADMDialog->GetItem(dialog, kDHeight_editText);
    sADMItem->SetUnits(item, kADMNoUnits);
    sADMItem->SetIntValue(item, (int)gHeight);
    sADMItem->SetMinIntValue(item, kHeightMin);
    sADMItem->SetMaxIntValue(item, kHeightMax);
    sADMItem->SetNotifyProc(item, DoHeightEditText);

    // Set up Resolution edit text:
    item = sADMDialog->GetItem(dialog, kDResolution_editText);
    sADMItem->SetUnits(item, kADMNoUnits);
    sADMItem->SetIntValue(item, (int)gResolution);
    sADMItem->SetMinIntValue(item, kResolutionMin);
    sADMItem->SetMaxIntValue(item, kResolutionMax);
    sADMItem->SetNotifyProc(item, DoResolutionEditText);

    // Set up Mode radio buttons:
    // Translate gMode to some useful number:
    int buttonHighlighted = kDMode_button_RGB;
    switch (gMode)
        {
        case classCMYKColorMode:
            buttonHighlighted = kDMode_button_CMYK;
            break;
        case classRGBColorMode:
        default:
            // We don't know what it is, so make it RGB:
            buttonHighlighted = kDMode_button_RGB;
            break;
        }

    // Now loop and turn off all buttons but the one that should
    // be on:
    for
        (
        int loop = kDMode_button_RGB;
        loop <= kDMode_button_CMYK;
```

```
        loop++
        )
        {
        item = sADMDialog->GetItem(dialog, loop);
        if (loop == buttonHighlighted)
            sADMItem->Activate(item, true);
        else
            sADMItem->Activate(item, false);
        }

    // Set up Fill radio buttons:
    // etc.
```

This function starts by initializing its error condition as no error. Then a variable `item` of the type `ADMItemRef` is declared. ADM expects all dialog objects to have their own native type so that it can easily keep track of everything. In this case, we are creating a reference to a type `ADMItem`. Other possible ADM types include: `ADMDialog`, `ADMDrawer`, `ADMList`, `ADMEntry`, `ADMNotifier`, `ADMTracker`, `ADMIcon`, `ADMImage`, ADMUserData, `ADMTimer`, `ADMActionMask`, and `ADMChar`. These are defined in the ADM documentation and listed in the `ADMTypes.h` file.

The next thing this function does is set up the Cancel button to notify our UI routine when it is clicked by the user. Since we want to be able to support the Photoshop convention of using the Cancel button as a Reset to Default Parameters function (when the user holds down the Option/Alt key when pressing Cancel), we must be notified when the Cancel button occurs. The call to `InitCancelButton(dialog)` sets up these parameters. `InitCancelButton()` will be discussed after `DoUIInit()`.

Next, we set up the edit text dialog boxes for Width, Height, and Resolution, as well as the Mode radio buttons. The various options (units, value, minimum value, maximum value) are handled using ADM function calls as shown here for the Width dialog.

```
    // Set up Width edit text:
    item = sADMDialog->GetItem(dialog, kDWidth_editText);
    sADMItem->SetUnits(item, kADMNoUnits);
    sADMItem->SetIntValue(item, (int)gWidth);
    sADMItem->SetMinIntValue(item, kWidthMin);
    sADMItem->SetMaxIntValue(item, kWidthMax);
    sADMItem->SetNotifyProc(item, DoWidthEditText);
```

First, it sets the item we are dealing with using the `sADMDialog->GetItem(dialog, kDWidth_editText)` function call. Then `SetUnit()` the various minimum, maximum, and current values, and finally `SetNotifyProc()` is used to track what is going on for each dialog option box. This function will notify our dialog routines whenever the user has changed something in the Width Edit Text box.

Notifiers are important ADM functions that are further explained in the ADM documentation. ADM reports a number of different user actions and provides this information via Notify routines. ADM notifiers track a number of user events: User Changed, Entry Text Changed, Close Hit

Notifier, Collapse Notifier, Expand Notifier, Bound Changed, Hide Window Modifier, etc. `ADMNotifier.h` defines all of the different ADM Notifiers that are available. The User Changed Notifier is helpful since it notifies our routines that the user has changed something in our dialog boxes.

The same type of ADM set up that we did on Width is performed for Height, and Resolution. The radio buttons are handled by a switch statement, and a simple loop is used to turn off all of the buttons except the one that is currently selected.

This completes our `DoUIInit()` routine.

We `InitCancelButton()` to receive notification when the user has pressed the Cancel button. We want to support the Photoshop convention of allowing dual use of the Cancel button. When the user presses option key while clicking Cancel, the function performed is reset values to default values and the Cancel button displays the Reset label instead of the Cancel label. Our dialog routines must be notified when this condition occurs.

`InitCancelButton()`takes a reference to the current dialog box and returns any error that occurs while initializing the Cancel button to trap the opotion/alt and mousedown notifiers and call our routines when these events happen.

```
static SPErr InitCancelButton(ADMDialogRef dialog)
    {
    SPErr error = kSPNoError;

    if (dialog != NULL)
        {
        // Set up "Cancel" button to notify us when its been clicked:
        ADMItemRef item = sADMDialog->GetItem(dialog, kDCancel_button);
        if (item != NULL)
            {
            sADMItem->SetNotifyProc(item, DoCancel);

            // Set up name of Cancel button.  Since we have to have
            // resources around for "Reset" and "Cancel", we might
            // as well check them to load the right value:
            if (gCancelButtonIsReset)
                SetTextToReset(item);
            else
                SetTextToCancel(item);

            // Set up mask for tracker function:
            ADMActionMask mask = sADMItem->GetMask(item);
            sADMItem->SetMask
                (
                item,
                mask |
                kADMModKeyDownMask |
                kADMModKeyUpMask |
                kADMLeaveMask |
                kADMEnterMask |
                kADMButtonUpMask
                );
```

```
            // Install tracker for it to be "Reset" when it needs to:
            sADMItem->SetTrackProc(item, DoReset);
            }
        else
            {
            error = kSPBadParameterError;
            }
        }
    else
        {
        error = kSPBadParameterError;
        }

    return error;

    }
```

The routine starts with a no error assigment. It then checks to make sure that the dialog is not NULL, and then it uses the ADM Dialog Suite:

```
ADMItemRef item = sADMDialog->GetItem(dialog, kDCancel_button);
```

to get a reference to the Cancel button of the current dialog. We then set up the notifier to call our routine when the button is pressed:

```
sADMItem->SetNotifyProc(item, DoCancel);
```

Since we can have two conditions for our Cancel button (Cancel and Reset) we set the text to be displayed on the button using `SetTextToCancel` or `SetTextToReset` functions depending upon which condition is true as specified by the state of our Boolean variable: `gCancelButtonIsReset`.

Finally, we must set up the mask for checking whether the Option/Alt key has been pressed in addition to the Cancel button being pressed. This mask information is set up using `SetMask()`.

You set up a mask to tell ADM what to track and when. This is done by declaring a mask, getting the current mask and then setting a new mask to track the additional events you want:

```
            ADMActionMask mask = sADMItem->GetMask(item);
            sADMItem->SetMask
                (
                item,
                mask |
                kADMModKeyDownMask |
                kADMModKeyUpMask |
                kADMLeaveMask |
                kADMEnterMask |
                kADMButtonUpMask
                );
```

To track the Option/Alt key event, we use a special type of notifier called a tracker. It also must be set up to call our routine when an event occurs:

```
sADMItem->SetTrackProc(item, DoReset);
```

We are choosing to track the Mod Key which is the Option key on the Macintosh and Alt key on the Windows platform up and down, and whether the mouse leaves or enters the Cancel button, and when the user actually clicks. This will let us control how the Cancel button turns into the Reset button (when the option key is pressed and the mouse is over the Cancel button) and when the reset is actually performed (upon click with the Option/Alt key pressed).

This routine is called when the cancel button is pressed. It is a notifier-receiver:

```
static void ASAPI DoCancel
    (
    ADMItemRefitem,
    ADMNotifierRefnotifier
    )
    {
    if (sADMNotify->IsNotifierType(notifier, kADMUserChangedNotifier))
        { // Correct notifier.  Do this:
        if (gCancelButtonIsReset)
            { // Must be reset!
            RestoreParameters(); // Resets.
            ADMDialogRef dialog = sADMItem->GetDialog(item);
            DoUIInit(dialog);
            }
        else
            {
            sADMItem->DefaultNotify(item, notifier);
            }
        }
    else
        {
        sADMItem->DefaultNotify(item, notifier);
        }
    }
```

`if (sADMNotify->IsNotifierType(notifier, kADMUserChangedNotifier))` determines what type of user input has been performed. This is possible because ADM tracks a number of different user actions and provides this information. `ADMNotifier.h` defines all of the different ADM Notifiers available. In our case, `DoCancel()` obtains the type of notifier and tests it to see if the Cancel button is “Reset” (using the `gCancelButtonIsReset` boolean variable defined at the top of the `MakeNewUI.cpp` file). If it is, then the previous parameters will be restored, and `DoUIInit()` is called to stuff the default parameters into their edit text fields and reset check and radio button groups.

The next routines in the `MakeNewUI.cpp` source file are `DoEditText()` for `Width`, `Height`, and `Resolution`.

The `DoEditText()` routines are called when the text is changed so that the value can be acquired and put in a global variable.

To acquire the new values in the dialog boxes, we use the ADM functions to get the “value” of the changed dialog box.

```
gWidth = sADMItem->GetIntValue(item);
```

This is another way that ADM simplifies the handling of our dialog boxes. In native code for the Mac (or PC) to obtain the value of the user input, we would have to acquire the string data, and then convert to integer values, etc. ADM is much more elegant (and easier!).

Finally, we use `SaveParameters()` and `RestoreParameters()` to save and restore our parameters.

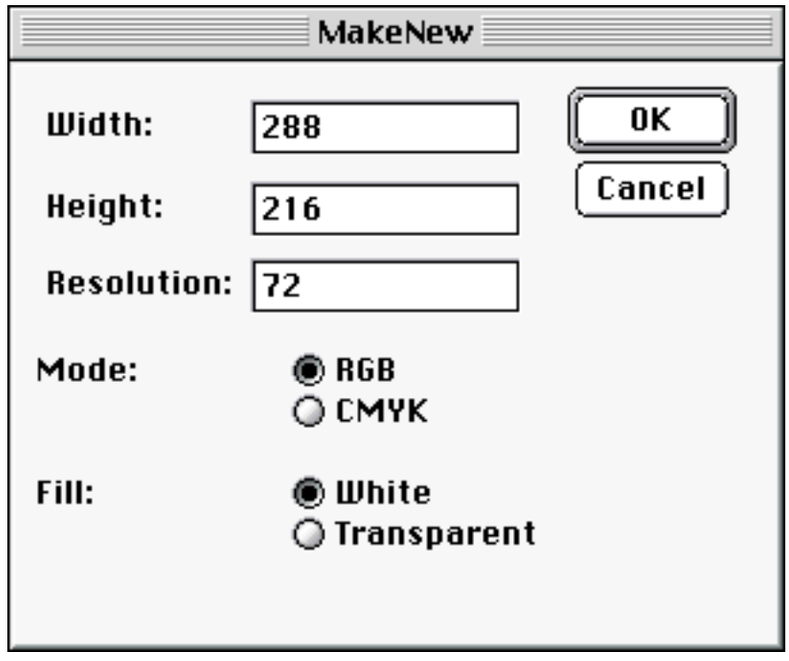
That’s it!

We’ve now revised all of the files (header, source, and resource files) associated with MakeNew and are ready to compile our project.

Step 14 - Compile the project using CodeWarrior or your development system. Find and correct any compile time errors.

Step 15 - Place the final `MakeNew.8li` binary file (our plug-in) into the Plug-ins folder in the Photoshop application folder.

Step 16 - Run Photoshop and select our plug-in from the Automation menu. Sit back and enjoy!



Other Good Information:

A Note on Error Types: There are several types of similar error types used in our plug-in, in Photoshop, in ADM and in the PICA suites. Generally, they are defined as follows:

Table 3–3:

Error	Type
OSErr	Photoshop uses the Operating System type errors
SPErr	PICA uses its own error types (Sweet Pea or SP)
ASErr	Equivalent to SPErr, can be considered an Adobe Systems error type (ASErr).

4. Using Listener

The purpose of this chapter is to give you a jumpstart in creating automation plug-ins using the Listener, a Photoshop automation plug-in. You need minimal experience in designing automation plug-ins. This is the fastest and simplest way of assembling a working automation plug-in. After going through the tutorial, you should be able to create more complex automation plug-ins. Listener is the key in automation plug-in design.

Listener Does Most Of The Work

The Listener plug-in automatically generates source code to help you. Itself an automation plug-in, Listener is included in the Sample Code folder of the Photoshop SDK. Once placed in the Photoshop Plug-Ins folder, Listener operates in the background during a working session. The Listener plug-in "listens" or generates code for all the "actionable events" - events that are scriptable. For example, Listener can record the selection of the paint tool, but can not record its application - the strokes that are created on a document, the color, nor the positioning on a document.

Listener Creates The Code

The Listener outputs C source code including- the declarations, initializations, and function calls, necessary to execute actionable events. The Listener outputs the code into a text file, named "Listener.log," which is created automatically. (Listener creates Listener.log file on the desktop for the Mac; in the C:\ drive for Windows.)

The code generated by the Listener plug-in resembles the following code:

```
void PlayeventMake(void)
{ PIActionDescriptor result;
     PIActionDescriptor desc00000B38;
     sPSActionDescriptor->Make(&desc00000B38);
          PIActionDescriptor desc00000B40;
          sPSActionDescriptor->Make(&desc00000B40);
          sPSActionDescriptor->PutClass(desc00000B40, keyMode,
                              classRGBColorMode);
          sPSActionDescriptor->PutUnitFloat(desc00000B40, keyWidth,
                              unitDistance, 576);
          sPSActionDescriptor->PutUnitFloat(desc00000B40, keyHeight,
                              unitDistance, 576);
          sPSActionDescriptor->PutUnitFloat(desc00000B40, keyResolution,
                              unitDensity, 72);
          sPSActionDescriptor->PutEnumerated(desc00000B40, keyFill, typeFill,
                              enumWhite);
     sPSActionDescriptor->PutObject(desc00000B38, keyNew, classDocument,
                              desc00000B40);
     sPSActionControl->Play(&result, eventMake, desc00000B38,
                              plugInDialogSilent);
}
```

How To Develop An Automation Plug-in

This section describes the steps to build an automation plug-in using the Listener plug-in. Within Photoshop, while the Listener is “listening” in the background, you will manually go through the actions that your plug-in would do. Next, you will copy the C code from the Listener.log file and paste it into an existing plug-in project. Compile the modified plug-in. Aside from possible syntactical errors, you will have created your own automation plug-in.

Step 1: Copy Listener Plug-in ToPhotoshop Plug-Ins Folder

The Photoshop SDK contains the complete source code for the Listener plug-in. The Listener plug-in is located in the Automation folder inside the Sample Code folder of the Photoshop SDK. Copy the Listener plug-in or create a shortcut to the Photoshop Plug-Ins folder. At launch, Photoshop loads the plug-ins from the Photoshop/Plug-Ins folder.

Step 2: Clear the Listener.log file

Make sure that your Listener.log file is empty, clear of any code that may have been recorded from previous Photoshop sessions. Close Photoshop. Open your Listener.log file in a text editor and delete its contents. From now on, whatever you do in Photoshop will be recorded by Listener.

Step 3: Prepare the Environment

Launch Photoshop and open a test document or a new document. Prepare the environment in which the plug-in will be used. Depending on the environment, certain menu items will be disabled or enabled. The Listener can only record events that involve actions that are enabled for that environment. Consider the document settings, such as color mode, background, dimensions, etc…Choose settings that will enable menu items that will be featured in your new plug-in. For this reason, it’s important to have a clear understanding of how and where your plug-in will be used. Keep in mind that the Listener plug-in is following every action happening in Photoshop. If there is an action that you don’t want to be included in the plug-in, you must edit it out of the Listener.log file.

Step 4: Apply Desired Actions On Test Document

Once you have the appropriate document opened and all the needed menu items enabled, apply all the actions to the document that you want your new automation plug-in to do. After you finish performing the intended sequence of events, open the Listener.log file. You will see a list of function definitions. The first function might be called “PlayeventOpen” or PlayeventNew

Step 5: Build Off An Existing Automation Plug-in: MakeNew

Once you have the code that will execute the desired series of actions, you will need to incorporate the code into an existing automation plug-in. The MakeNew sample plug-in project (located in the Automation folder inside the Sample Code folder) has all the components of an automation plug-in. The MakeNew plug-in creates a new Photoshop document, which is no different than File->New. You simply need to paste in the coding that executes the actions.

Step 6: Verify The Contents Of Listener.log File

Open your Listener.log file. There should be code that resembles the following:

```
void PlayeventNotify(void)
{
     PIActionDescriptor result;
     PIActionDescriptor desc000001A8;
     sPSActionDescriptor->Make(&desc000001A8);
     sPSActionDescriptor->PutEnumerated(desc000001A8, keyWhat,
                              typeNotify, enumFirstIdle);
     sPSActionControl->Play(&result, eventNotify, desc000001A8,
                              plugInDialogSilent);
}

void PlayeventMake(void)
{
     PIActionDescriptor result;
     PIActionDescriptor desc000001D0;
     sPSActionDescriptor->Make(&desc000001D0);
          PIActionDescriptor desc000001D8;
          sPSActionDescriptor->Make(&desc000001D8);
          sPSActionDescriptor->PutClass(desc000001D8, keyMode,
                              classRGBColorMode);
          sPSActionDescriptor->PutUnitFloat(desc000001D8, keyWidth,
                              unitDistance, 1024);
          sPSActionDescriptor->PutUnitFloat(desc000001D8, keyHeight,
                              unitDistance, 768);
          sPSActionDescriptor->PutUnitFloat(desc000001D8, keyResolution,
                              unitDensity, 72);
          sPSActionDescriptor->PutEnumerated(desc000001D8, keyFill,
                              typeFill, enumTransparent);
     sPSActionDescriptor->PutObject(desc000001D0, keyNew,
                              classDocument, desc000001D8);
     sPSActionControl->Play(&result, eventMake, desc000001D0,
                              plugInDialogSilent);
}

void PlayeventSelect(void)
{
     PIActionDescriptor result;
     PIActionDescriptor desc00000248;
     sPSActionDescriptor->Make(&desc00000248);
          PIActionReference ref00000050;
          sPSActionReference->Make(&ref00000050);
          sPSActionReference->PutClass(ref00000050, classPaintbrushTool);
     sPSActionDescriptor->PutReference(desc00000248, keyNull, ref00000050);
     sPSActionControl->Play(&result, eventSelect, desc00000248,
                              plugInDialogSilent);
}
```

Notice that in the above sample code, there are three separate functions: PlayeventNotify, PlayeventMake and PlayeventSelect. Depending on how many actions you want to automate, the code in your Listener.log file

may have more and/or different functions. The PlayeventMake function creates a new document. PlayeventSelect selects one of the tools in the toolbar. PlayeventNotify is an action that occurs at launch of Photoshop. Most likely your Listener output contains the PlayeventNotify and PlayeventMake or PlayeventOpen. These functions are not necessary in the plug-in you want to create. You don't want your plug-in to create a document or open the same document every time its invoked. Therefore, you will need to delete the PlayeventNotify and PlayeventMake or PlayeventOpen functions.

Step 7: Incorporate Function Calls

You must incorporate the functions of the desired actions into the MakeNew plug-in. Modify MakeNew.cpp to add function prototypes and C function calls in the appropriate places. In the MakeNew source file, MakeNew.cpp, the Execute function is the main function. Place the respective function calls for the desired actions, as specified by "***function calls go here***" in the following excerpt from MakeNew.cpp:

```
error = DoUI();

***function calls go here***

if (error ==kSPNoError)
{
MakeNewDocument();
WriteScriptParams();
}
```

To demonstrate using the previous sample code, the modified code would look like the following:

```
error = DoUI();

PlayeventMake();
PlayeventSet();

if (error ==kSPNoError)
{
MakeNewDocument();
WriteScriptParams();
}
```

Step 8: Paste Function Definitions and Function Prototypes

Listener.log file consists entirely of a list of function definitions. In the Listener.log file, "Select All" text and "Paste" it at the end of the MakeNew.cpp file.

Next, include their appropriate prototypes in the "Prototypes" section at the beginning of MakeNew.cpp.

For example, for the previous sample code, the prototypes would be:

```
void PlayeventMake(void);
void PlayeventSet(void);
```

Step 9: Comment Out The MakeNew Actions

In the MakeNew.cpp file, the Execute function controls the actions in MakeNew. You need to remove or comment out the code that characterize the MakeNew plug-in. First, remove the DoUI() function calls. This function pops the MakeNew dialog, which is not necessary for your plug-in. You will also comment out MakeNewDocument function call which performs all the MakeNew actions. The WriteScriptParams features the scripting for MakeNew. We will modify the WriteScriptParams to display the parameters of your plug-in in proceeding sections.

Remove the appropriate lines of code, as illustrated in the following code:

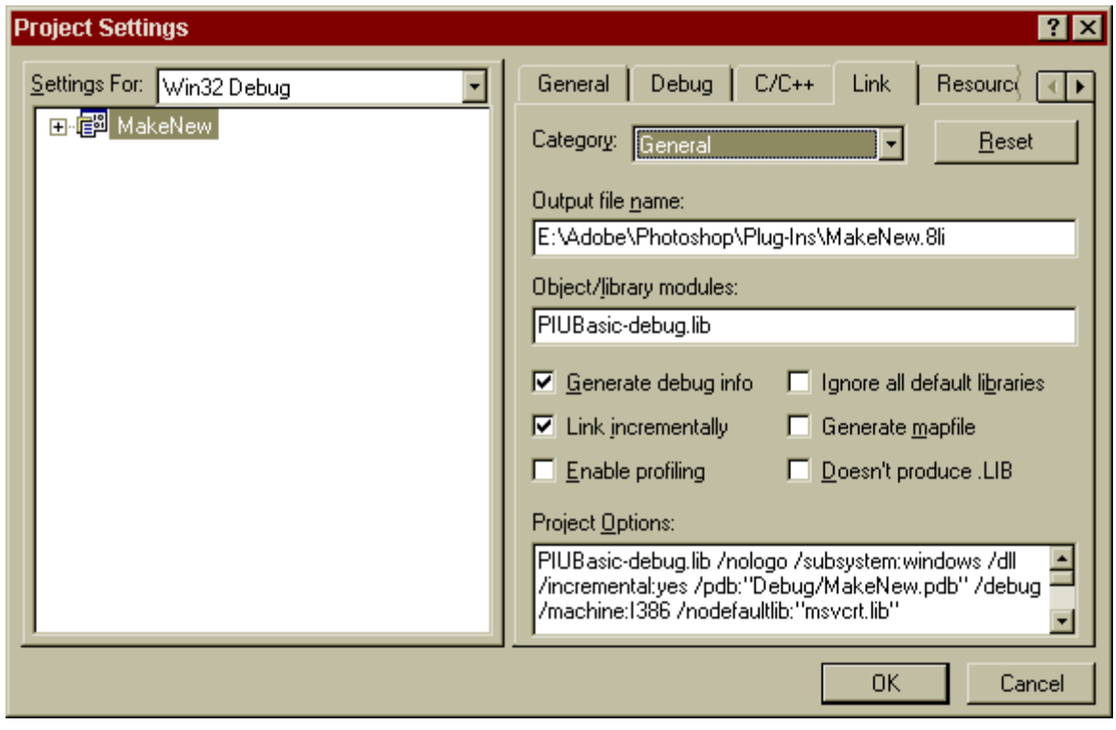
```
//error = DoUI();

***function calls go here***

//if (error ==kSPNoError)
//{
//MakeNewDocument();
//WriteScriptParams();
//}
```

Step 10: Build Your Plug-in And You Are Done!

Now, you have made the necessary code changes for your new plug-in. Before building, make sure that under Project Settings, the build is outputted to the Photoshop Plug-Ins folder. Rebuild the MakeNew plug-in. After a successful build, close Photoshop and launch it again, so it will



reload all its plug-ins, including your new plug-in. Once in Photoshop, open a test document, and invoke your plug-in by selecting MakeNew under File->Automation->MakeNew.

Congratulations! You have just made your first automation plug-in!

note: If you want to enable scripting or create a UI for your plug-in refer to Chapter 3 for details.

5. More Listener

At this point, you know how to use Listener to create code for you. Unfortunately, Listener has its limitations. This chapter explains the Listener output in detail and the code needed to fill in the holes of Listener output.

Type Tool

Listener cannot retrieve type tool data from Photoshop. Listener will always give you a pointer called voidPtrToData. The PlayeventMake function containing voidPtrData will look similar to the following code:

```
void PlayeventMake(void)
{
     PIActionDescriptor result;
     PIActionDescriptor desc00000200;
     sPSActionDescriptor->Make(&desc00000200);
          PIActionReference ref00000048;
          sPSActionReference->Make(&ref00000048);
          sPSActionReference->PutClass(ref00000048, classTextLayer);
     sPSActionDescriptor->PutReference(desc00000200, keyNull, ref00000048);
          PIActionDescriptor desc00000208;
          sPSActionDescriptor->Make(&desc00000208);
         sPSActionDescriptor->PutString(desc00000208, keyText, "Hello World!");
          sPSActionDescriptor->PutData(desc00000208, keyTextData, 143,
                              voidPtrToData);
               PIActionDescriptor desc00000210;
               sPSActionDescriptor->Make(&desc00000210);
               sPSActionDescriptor->PutUnitFloat(desc00000210, keyHorizontal,
                              unitPercent, 22.3999);
               sPSActionDescriptor->PutUnitFloat(desc00000210, keyVertical,
                              unitPercent, 33.9152);
          sPSActionDescriptor->PutObject(desc00000208, keyTextClickPoint,
                              classPoint, desc00000210);
     sPSActionDescriptor->PutObject(desc00000200, keyUsing, classTextLayer,
                              desc00000208);
     sPSActionControl->Play(&result, eventMake, desc00000200,
                              plugInDialogSilent);
}
```

Replace the PlayeventMake in your Listener output with the MakeTypeLayer function. Be sure to include the desired parameters, such as HorizontalPosition=29; VerticalPosition=40; Text = "hello world"; FontName = "Helvetica"; Alignment = enumLeft; Size = 12;antialias = true; AutoKern = true;

```
void MakeTypeLayer(double HorizontalPosition, double VerticalPosition , char *
Text, char *FontName, long Alignment, double Size, bool antialias, bool AutoK-
ern)
{
PIActionDescriptor descriptor;
PIActionDescriptor positionDescriptor;
PIActionDescriptor typeDescriptor;
PIActionDescriptor result;
PIActionReference reference;

sPSActionDescriptor->Make(&descriptor);
sPSActionDescriptor->Make(&positionDescriptor);
sPSActionDescriptor->Make(&typeDescriptor);
```

```
sPSActionDescriptor->Make(&result);
sPSActionReference->Make(&reference);

sPSActionReference->PutClass(reference, classTextLayer);
sPSActionDescriptor->PutReference(descriptor, keyNull, reference);
sPSActionDescriptor->PutUnitFloat(positionDescriptor, keyHorizontal,
      unitDistance, HorizontalPosition);
sPSActionDescriptor->PutUnitFloat(positionDescriptor, keyVertical,
      unitDistance, VerticalPosition);
sPSActionDescriptor->PutObject(typeDescriptor, keyTextClickPoint,
      classPoint, positionDescriptor);
sPSActionDescriptor->PutString(typeDescriptor, keyText, Text);
sPSActionDescriptor->PutString(typeDescriptor, keyFontName, FontName);
sPSActionDescriptor->PutInteger(typeDescriptor, keyScript, 0);
sPSActionDescriptor->PutEnumerated(typeDescriptor, keyAlignment,
      typeAlignment, Alignment);
sPSActionDescriptor->PutInteger(typeDescriptor, keySize, Size);
sPSActionDescriptor->PutBoolean(typeDescriptor, keyAntiAlias, antialias);
sPSActionDescriptor->PutBoolean(typeDescriptor, keyAutoKern, AutoKern);
//to get vertical text:
sPSActionDescriptor->PutBoolean( typeDescriptor, keyVertical, true );
sPSActionDescriptor->PutBoolean(typeDescriptor,keyRotate,true);

sPSActionDescriptor->PutObject(descriptor, keyUsing, classTextLayer,
      typeDescriptor);
sPSActionControl->Play(&result, eventMake, descriptor, plugInDialogSilent);
};
```

6. Filter Action Events

This chapter describes of all the actionable filter events that are available in Photoshop. Some filters are internal to Photoshop and cannot be removed from the program. Others are included in the Plug-in folder and can be removed from the program by removing them from the Plug-in folder. This chapter details those two type of filters: internal filters and plug-in filters.

Filter Events are events that modify the pixels of a document. They are most commonly found in the Filter menu.

In Photoshop itself, you are limited to the values you can input into any given filter by the pinned values defined in the UI. When programming automation plug-ins, you are unrestricted in the values you may enter, and it is often good practice to pin values in your code to avoid Photoshop coming back with errors.

After the event name, a table will list the required parameters for the descriptor.

Unless a specific target is given, the action will be performed on the current element up the current target chain that is capable of performing that action.

Built-In Filter Events

eventAddNoise ('AdNs')

Image Reference

Table 4–4: eventAddNoise Parameters (3)

Key	Type	Bounds	Options
keyAmount ('Amnt')	typeInteger	1 .. 999	flagsSingleParameter
keyDistribution ('Dstr')	typeDistribution ('Dstr')		flagsEnumeratedParameter
keyMonochromatic ('Mnch')	typeBoolean		flagsSingleParameter

eventBlur ('Blr ')

Image Reference

takes no parameters

eventBlurMore ('BlrM')

Image Reference

takes no parameters

eventBorder ('Brdr')

Image Reference

Table 4–5: eventBorder Parameter (1)

Key	Type	Bounds	Options
keyWidth (‘Wdth')	unitfloat /unit pixels ('#Pxl')		flagsEnumeratedParameter

keyWidth
This is the amount of units to expand away from the current selection path. The units available are `pixels` and `distance`.

eventBrightness ('BrgC')

Image Reference

Table 4–6: eventBrightness Parameters (2)

Key	Type	Bounds	Options
keyBrightness ('Brgh')	typeInteger	-100..100	flagsOptionalSingleParameter
keyContrast ('Cntr')	typeInteger	-100..100	flagsOptionalSingleParameter

eventColorBalance ('ClrB')

Image Reference

Table 4–7: eventColorBalance Parameters (4)

Key	Type	Bounds	Options
keyShadowLevels ('ShdL')	typeInteger		flagsListParameter
keyMidtoneLevels ('MdtL')	typeInteger		flagsListParameter
keyHighlightLevels ('HghL')	typeInteger		flagsListParameter
keyPreserveLuminosity ('PrsL')	typeBoolean		flagsOptionalSingleParameter

targets: null

keyShadowLevels
This is the percentage amount by which the Cyan/Red, Magenta/Green, and Yellow/Blue sliders are changed.

keyMidtoneLevels
This is the percentage amount by which the Cyan/Red, Magenta/Green, and Yellow/Blue sliders are changed.

keyHighlightLevels
This is the percentage amount by which the Cyan/Red, Magenta/Green, and Yellow/Blue sliders are changed.

keyPreserveLuminosity
This toggles whether or not the luminosity channel of the image is to be preserved. True means yes.

eventColorRange ('ClrR')

Image Reference

Table 4–8: eventColorRange Parameters (6)

Key	Type	Bounds	Options
keyUsing ('Usng')	typeFSS		flagsOptionalSingleParameter
keyFuzziness ('Fzns')	typeInteger	0 . . 200	flagsOptionalSingleParameter
keyMinimum ('Mnm ')	classColor('Clr ')		flagsOptionalSingleParameter
keyMaximum ('Mxm ')	classColor ('Clr ')		flagsOptionalSingleParameter
keyColors ('Clrs')	typeColors ('Clrs')		flagsOptionalEnumeratedParameter
keyInvert ('Invr')	typeBoolean		flagsOptionalSingleParameter

eventContract ('Cntc')

Image Reference

Table 4–9: eventContract Parameter (1)

Key	Type	Bounds	Options
keyBy ('By ')	unitFloat/unitPixels ('#Pxl')		flagsEnumeratedParameter

targets: null

keyBy
This is the number in pixels by which the selection mask will shrink.

eventCurves ('Crvs')

Image Reference

Table 4–10: eventCurves Parameters (2)

Key	Type	Bounds	Options
keyUsing ('Usng')	typeFSS		flagsOptionalSingleParameter
keyAdjustment ('Adjs')	classCurvesAdjustment ('CrvA')		flagsOptionalListParameter

eventCustom ('Cstm')

Image Reference

Table 4–11: eventCustom Parameters (3)

Key	Type	Bounds	Options
keyMatrix ('Mtrx')	typeInteger		required, listOfItems, notEnumerated, reserved, reserved, reserved, reserved, reserved,reserved, reserved, reserved, reserved, prepositionParam, notFeminine, notMasculine, singular
keyScale ('Scl ')	typeInteger		flagsSingleParameter
keyOffset ('Ofst')	typeInteger		flagsSingleParameter

eventDefringe ('DfnP')

Image Reference

Table 4–12: eventDefringe Parameter (1)

Key	Type	Bounds	Options
keyWidth ('Wdth')	unitFloat/unitPixels ('#Pxl')		flagsEnumeratedParameter

targets: null

keyWidth
This is the amount by which target will be defringed.

eventDesaturate ('Dstt')

Image Reference

takes no parameters
targets: null

eventDespeckle ('Dspc')

Image Reference

takes no parameters
targets: null

eventDiffuse ('Dfs ')

Image Reference

Table 4–13: eventDiffuse Parameter (1)

Key	Type	Bounds	Options
eventMode ('Md ')	typeDiffuseMode ('DfsM')		flagsEnumeratedParame-ter

eventDustAndScratches ('DstS')

Image Reference

Table 4–14: eventDustAndScratches Parameters (2)

Key	Type	Bounds	Options
keyRadius ('Rds ')	typeInteger	1 . . 16	flagsSingleParameter
keyThreshold ('Thsh')	typeInteger	0 . . 255	flagsSingleParameter

eventEmboss ('Embs')

Image Reference

Table 4–15: eventEmboss Parameters (3)

Key	Type	Bounds	Options
keyAngle ('Angl')	typeInteger	-360 . ..360	flagsSingleParameter
keyHeight ('Hght')	typeInteger	1 . . 10	flagsSingleParameter
keyAmount ('Amnt')	typeInteger	1 . . 500	flagsSingleParameter

targets: null

eventEqualize ('Eqlz')

Image Reference

Table 4–16: eventEqualize Parameter (1)

Key	Type	Bounds	Options
keyArea ('Area')	typeAreaSelector ('ArSl')		flagsOptionalEnumerated-Parameter

eventExpand ('Expn')

Image Reference

Table 4–17: eventExpand Parameter (1)

Key	Type	Bounds	Options
keyBy ('By ')	unitFloat/unitPixels ('#Pxl')		flagsOptionalEnumerated-Parameter

eventFacet ('Fct ')

Image Reference

takes no parameters
targets: null

eventFade ('Fade')

Image Reference

Table 4–18: eventFade Parameters (2)

Key	Type	Bounds	Options
keyOpacity ('Opct')	unitFloat/unitPercent ('#Prc')		flagsEnumeratedParame-ter
keyMode ('Md ')	typeBlendMode ('BldM')		flagsEnumeratedParame-ter

eventFeather ('Fthr')

Image Reference

Table 4–19: eventFeather Parameter (1)

Key	Type	Bounds	Options
keyRadius ('Fthr')	unitFloat/unitPixels ('#Pxl')		flagsEnumeratedParame-ter

targets: null

keyRadius
This is the amount by which the target will be feathered. The number represents the radius by which the Gaussian Bell Curve is applied to the selection mask.

eventFill ('Fl ')

Image Reference

Table 4–20: eventFill Parameters (10)

Key	Type	Bounds	Options
keyFrom('From')	typeLocationReference ('#Lct')		flagsSingleParameter
keyUsing ('Usng')	typeFillContents ('FlCn')		flagsEnumeratedParame-ter
keyOpacity ('Opct')	unitFloat/unitPercent ('#Prc')		flagsEnumeratedParame-ter

Table 4–20: eventFill Parameters (10)

Key	Type	Bounds	Options
keyMode ('Md ')'	typeBlendMode ('BldM')		flagsEnumeratedParameter
keyPreserveTransparency ('PrsT')	typeBoolean		flagsOptionalEnumeratedParameter
keyMerged ('Mrgd')	typeBoolean		flagsOptionalSingleParameter
keyTolerance ('Tlrn')	typeInteger		flagsSingleParameter
keyRadius ('Rds ')	unitFloat/unitPixels ('#Pxl')		flagsOptionalEnumeratedParameter
keyAntiAlias ('AntA')	typeBoolean		flagsOptionalEnumeratedParameter
keyWholePath ('WhPt')	typeBoolean		flagsOptionalEnumeratedParameter

eventFilter ('Fltr')

Image Reference

Table 4–21: eventFilter Parameters (2)

Key	Type	Bounds	Options
keyUsing ('Usng'')	typeChar		flagsSingleParameter
keyDatum ('Dt ')	typeChar		optional, singleItem, notEnumerated, reserved, reserved,reserved, reserved, reserved,reserved, reserved, reserved, reserved,prepositionParam, notFeminine, notMasculine, singular

eventFindEdges ('FndE')

Image Reference

takes no parameters
targets: null

eventFragment ('Frgm')

Image Reference

takes no parameters
targets: null

eventGaussianBlur ('GsnB')

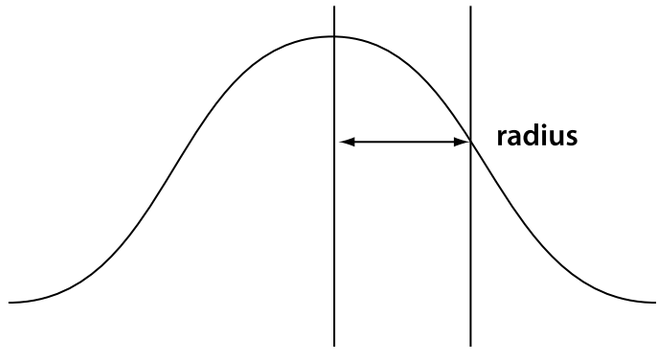
Image Reference

Table 4–22: eventGaussianBlur Parameter (1)

Key	Type	Bounds	Options
keyRadius ('Rds ''	typeFloat	0.1 . . 250	flagsSingleParameter

targets: null

keyRadius
This parameter specifies the radius of the circle drawn away from the center of the Gauusian Bell Curve, and every pixel is subjected to this bell curve, with the pixel in question at the peak.



eventGrow ('Grow')

Image Reference

takes no parameters
targets: null

eventHighPass ('HghP')

Image Reference

Table 4–23: eventHighPass Parameter (1)

Key	Type	Bounds	Options
keyRadius ('Rds ')	typeFloat	0.1 . . 250	flagsSingleParameter

eventHueSaturation ('HStr')

Image Reference

Table 4–24: eventHueSaturation Parameters (3)

Key	Type	Bounds	Options
keyColorize ('Clrz')	typeBoolean		flagsSingleParameter
keyUsing ('Usng')	typeFSS		flagsOptionalSingleParameter
keyAdjustment ('Adjs')	classHueSatAdjustment ('HStA')		flagsOptionalListParameter

eventInverse ('ImgS')

Image Reference

takes no parameters
targets: null

eventInvert ('Invr')

Image Reference

takes no parameters
targets: null

eventLevels ('Lvls')

Image Reference

Table 4–25: eventLevels Parameters (3)

Key	Type	Bounds	Options
keyUsing ('Usng')	typeFSS		flagsOptionalSingleParameter
keyAuto ('Auto')	typeBoolean		flagsOptionalSingleParameter
keyAdjustment ('Adjs')	classLevelsAdjustment ('LvlA ')		flagsOptionalListParameter

eventMaximum ('Mxm ')

Image Reference

Table 4–26: eventMaximum Parameter (1)

Key	Type	Bounds	Options
keyRadius ('Rds ')	typeInteger	1 . . 16	flagsSingleParameter

eventMedian ('Mdn ')

Image Reference

Table 4–27: eventMedian Parameter (1)

Key	Type	Bounds	Options
keyRadius ('Rds ')	typeInteger	1 . . 16	flagsSingleParameter

eventMinimum ('Mnm ')

Image Reference

Table 4–28: eventMinimum Parameter (1)

Key	Type	Bounds	Options
keyRadius ('Rds ')	typeInteger	1 . . 10	flagsSingleParameter

eventMosaic ('Msc ')

Image Reference

Table 4–29: eventMosaic Parameter (1)

Key	Type	Bounds	Options
keyCellSize ('ClSz ')	unitFloat/unitPixels ('#Pxl')	2 . . 64	flagsSingleParameter

eventMotionBlur ('MtnB')

Image Reference

Table 4–30: eventMotionBlur Parameters (2)

Key	Type	Bounds	Options
keyAngle ('Angl ')	typeInteger	-90 . . 90	flagsSingleParameter
keyDistance ('Dstn')	typeInteger	1 . . 999	flagsSingleParameter

eventOffset ('Ofst')

Image Reference

Table 4–31: eventOffset Parameters (3)

Key	Type	Bounds	Options
keyHorizontal ('Hrzn ')	typeInteger	-30000 . . 30000	flagsSingleParameter
keyVertical ('Vrtc')	typeInteger	-30000 . . 30000	flagsSingleParameter
keyFill ('Fl ')	typeFillMode ('FlMd')		flagsEnumeratedParameter

eventPosterize ('Pstr')

Image Reference

Table 4–32: eventPosterize Parameter (1)

Key	Type	Bounds	Options
keyLevels ('Lvls')	typeInteger	-90 . . 90	flagsSingleParameter

eventPurge ('Prge')

Image Reference

takes no parameters

eventReplaceColor ('RplC')

Image Reference

Table 4–33: eventReplaceColor Parameters (7)

Key	Type	Bounds	Options
keyUsing ('Usng')	typeFSS		flagsOptionalSingleParameter
keyFuzziness ('Fzns')	typeInteger		flagsSingleParameter
keyMinimum ('Mnm ')	classColor ('Clr ')		flagsSingleParameter
keyMaximum ('Mxm ')	classColor ('Clr ')		flagsSingleParameter
keyHue ('H ')	typeInteger		flagsOptionalSingleParameter
keySaturation ('Strt')	typeInteger		flagsOptionalSingleParameter
keyLightness ('Lght')	typeInteger		flagsSingleParameter

eventSelectiveColor ('SlcC')

Image Reference

Table 4–34: eventSelectiveColor Parameters (3)

Key	Type	Bounds	Options
keyUsing ('Usng')	typeFSS		flagsOptionalSingleParameter
keyColorCorrection ('ClrC')	classColorCorrection ('ClrC')		flagsListParameter
keyMethod ('Mthd')	typeCorrectionMethod ('CrcM')		flagsEnumeratedParameter

eventSharpen ('Shrp')

Image Reference

takes no parameters
targets: null

eventSharpen Edges ('ShrE')

Image Reference

takes no parameters
targets: null

eventSharpenMore ('ShrM')

Image Reference

takes no parameters

targets: null

eventSimilar ('Smlr')

Image Reference

takes no parameters
targets: null

eventSmooth ('Smth')

Image Reference

Table 4–35: eventSmooth Parameter (1)

Key	Type	Bounds	Options
keyradius ('Rds ')	unitFloat/unitPixels ('#Pxl')		flagsEnumeratedParameter

eventStroke ('Strk')

Image Reference

Table 4–36: eventStroke Parameters (6)

Key	Type	Bounds	Options
keyWidth ('Wdth')	typeInteger		flagsSingleParameter
keyLocation ('Lctn')	typeStrokeLocation ('StrL')		flagsEnumeratedParameter
keyOpacity ('Opct')	unitFloat/unitPercent ('#Prc')		flagsEnumeratedParameter
keyMode ('Md ')	typeBlendMode ('BldM')		flagsEnumeratedParameter
keyPreserveTransparency ('PrsT')	typeBoolean		flagsOptionalEnumeratedParameter
keyUsing ('Usrs')	typeClass ('Type')		flagsOptionalSingleParameter

eventThreshold ('Thrs')

Image Reference

Table 4–37: eventThreshold Parameter (1)

Key	Type	Bounds	Options
keyLevel ('Lvl ')	typeInteger	1 . . 255	flagsSingleParameter

eventTraceContour ('TrcC')

Image Reference

Table 4–38: eventTraceContour Parameters (2)

Key	Type	Bounds	Options
keyLevel ('Lvl ')	typeInteger	0 . . 255	flagsSingleParameter
keyEdge ('Edg ')	typeContourEdge ('CntE')		flagsEnumeratedParameter

eventTrap ('Trap')

Image Reference

Table 4–39: eventTrap Parameter (1)

Key	Type	Bounds	Options
keywidth ('Wdth')	unitFloat/unitDistance('#Rlt')		flagsEnumeratedParameter

eventUnsharpMask ('UsnM')

Image Reference

Table 4–40: eventUnsharpMask Parameters (3)

Key	Type	Bounds	Options
keyRadius ('Rds ')	typeFloat	1 . . 500	flagsSingleParameter
keyAmount ('Amnt')	typeInteger	0.1 . . 250	flagsSingleParameter
keyThreshold ('Thsh')	typeInteger	1 . . 255	flagsSingleParameter

Plug-in Filter Events

eventAccentedEdges ('AccE')

Image Reference

Table 4–41: eventAccentedEdge Parameters (3)

Key	Type	Bounds	Options
keyEdgeWidth ('EdgW ')	typeInteger	1..14	flagsSingleParameter
keyEdgeBrightness ('EdgB')	typeInteger	0..50	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..15	flagsSingleParameter

eventAngledStrokes ('AngS')

Image Reference

Table 4–42: eventAngledStrokes Parameters (3)

Key	Type	Bounds	Options
keyDirectionBalance ('DrcB ')	typeInteger	0..100	flagsSingleParameter
keyStrokeLength ('StrL')	typeInteger	3..50	flagsSingleParameter
keySharpness ('Shrp')	typeInteger	0..10	flagsSingleParameter

eventBasRelief ('BsRl')

Image Reference

Table 4–43: eventBasRelief Parameters (3)

Key	Type	Bounds	Options
keyDetail ('Dtl ')	typeInteger	0..15	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..15	flagsSingleParameter
keyLightDirection ('LghD')	typeLightDirection		flagsEnumeratedParame-ter

eventChalkCharcoal('ChlC')

Image Reference

Table 4–44: eventBasRelief Parameters (3)

Key	Type	Bounds	Options
keyCharcoalArea ('ChrA')	typeInteger	0..20	flagsSingleParameter
keyChalkArea ('ChlA')	typeInteger	1..20	flagsSingleParameter
keyStrokePressure ('StrP')	typeLightDirection	0..5	flagsSingleParameter

eventCharcoal ('Chrc')

Image Reference

Table 4–45: eventCharcoal Parameters (3)

Key	Type	Bounds	Options
keyCharcoalAmount ('ChAm')	typeInteger	0..7	flagsSingleParameter
keyDetail ('Dtl ')	typeInteger	0..5	flagsSingleParameter
keyLightDark ('LgDr')	typeInteger	0..100	flagsSingleParameter

eventChrome ('Chrm')

Image Reference

Table 4–46: eventChrome Parameters (2)

Key	Type	Bounds	Options
keyDetail ('Dtl ')	typeInteger	1..10	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..10	flagsSingleParameter

eventColoredPencil ('ClrP')

Image Reference

Table 4–47: eventColoredPencil Parameters (3)

Key	Type	Bounds	Options
keyPencilWidth ('Pncl')	typeInteger	1..24	flagsSingleParameter
keyStrokePressure ('StrP')	typeInteger	1..15	flagsSingleParameter
keyPaperBrightness ('PprB')	typeInteger	1..50	flagsSingleParameter

eventConteCrayon ('CntC')

Image Reference

Table 4–48: eventConteCrayon Parameters (8)

Key	Type	Bounds	Options
keyForegroundLevel ('FrgL')	typeInteger	1..15	flagsSingleParameter
keyBackgroundLevel ('BckL')	typeInteger	1..15	flagsSingleParameter
keyLightDirection ('LghD')	typeLightDirection		flagsEnumeratedParameter
keyTextureType ('TxtT')	typeTextureType		flagsOptionalSingleParameter
keyTextureType ('TxtT')	typeAlias		flagsOptionalSingleParameter
keyScaling ('Scln')	typeInteger	50..200	flagsSingleParameter
keyRelief ('Rlf ')	typeInteger	0..50	flagsSingleParameter
keyInvertTexture ('InvT')	typeBoolean		flagsSingleParameter

eventCraquelure ('Crql')

Image Reference

Table 4–49: eventCraquelure Parameters (3)

Key	Type	Bounds	Options
keyCrackSpacing ('CrcS')	typeInteger	2..100	flagsSingleParameter
keyCrackDepth ('CrcD')	typeInteger	1..15	flagsSingleParameter
keyCrackBrightness ('CrcB')	typeInteger	1..10	flagsSingleParameter

eventCrosshatch ('Crsh')

Image Reference

Table 4–50: eventCrosshatch Parameters (3)

Key	Type	Bounds	Options
keyStrokeLength ('StrL')	typeInteger	3..50	flagsSingleParameter
keySharpness ('Shrp')	typeInteger	0..20	flagsSingleParameter
keyStrength_PLUGIN ('Strg')	typeInteger	1..3	flagsSingleParameter

eventCutout ('Ct ')

Image Reference

Table 4–51: eventCutout Parameters (3)

Key	Type	Bounds	Options
keyNumberOfLayers ('NmbL')	typeInteger	2..8	flagsSingleParameter
keyEdgeSimplicity ('EdgS')	typeInteger	0..10	flagsSingleParameter
keyEdgeFidelity ('EdgF')	typeInteger	1..3	flagsSingleParameter

eventDarkStrokes ('DrkS')

Image Reference

Table 4–52: eventDarkStrokes Parameters (3)

Key	Type	Bounds	Options
keyBalance ('Blnc')	typeInteger	0..10	flagsSingleParameter
keyBlackIntensity ('Blcl')	typeInteger	0..10	flagsSingleParameter
keyWhiteIntensity ('Whtl')	typeInteger	0..10	flagsSingleParameter

eventDiffuseGlow ('DfsG')

Image Reference

Table 4–53: eventDiffuseGlow Parameters (3)

Key	Type	Bounds	Options
keyGraininess ('Grns')	typeInteger	0..10	flagsSingleParameter
keyGlowAmount ('GlwA')	typeInteger	0..20	flagsSingleParameter
keyClearAmount ('ClrA')	typeInteger	0..20	flagsSingleParameter

eventDryBrush ('DfsG')

Image Reference

Table 4–54: eventDryBrush Parameters (3)

Key	Type	Bounds	Options
keyBrushSize ('BrsS')	typeInteger	0..10	flagsSingleParameter
keyBrushDetail ('BrsD')	typeInteger	0..20	flagsSingleParameter
keyTexture ('Txtr')	typeInteger	1..3	flagsSingleParameter

eventFilmGrain ('FlmG')

Image Reference

Table 4–55: eventFilmGrain Parameters (3)

Key	Type	Bounds	Options
keyGrain ('Grn ')	typeInteger	0..20	flagsSingleParameter
keyHighlightArea ('HghA')	typeInteger	0..20	flagsSingleParameter
keyIntensity ('Intn')	typeInteger	0..20	flagsSingleParameter

eventFresco ('Frsc')

Image Reference

Table 4–56: eventFresco Parameters (3)

Key	Type	Bounds	Options
keyBrushSize ('BrsS')	typeInteger	0..10	flagsSingleParameter
keyBrushDetail ('BrsD')	typeInteger	0..10	flagsSingleParameter
keyTexture ('Txtr')	typeInteger	1..3	flagsSingleParameter

eventGlass ('Gls ')

Image Reference

Table 4–57: eventGlass Parameters (6)

Key	Type	Bounds	Options
keyDistortion ('Dstr')	typeInteger	0..20	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..15	flagsSingleParameter
keyTextureType ('TxtT')	typeTextureType		flagsOptionalSingleParameter
keyTextureType ('TxtT')	typeAlias		flagsOptionalSingleParameter
keyScaling ('Scln')	typeInteger	50..200	flagsSingleParameter
keyInvertTexture ('InvT')	typeBoolean		flagsSingleParameter

eventGlowingEdges ('GlwE')

Image Reference

Table 4–58: eventGlowingEdges Parameters (3)

Key	Type	Bounds	Options
keyEdgeWidth ('EdgW')	typeInteger	0..14	flagsSingleParameter
keyEdgeBrightness ('EdgB')	typeInteger	0..20	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..15	flagsSingleParameter

eventGrain ('Grn ')

Image Reference

Table 4–59: eventGrain Parameters (3)

Key	Type	Bounds	Options
keyIntensity ('Intn')	typeInteger	0..100	flagsSingleParameter
keyContrast ('Cntr')	typeInteger	0..100	flagsSingleParameter
keyGrainType ('Grnt')	typeGrainType		flagsEnumeratedParameter

eventGraphicPen ('GraP')

Image Reference

Table 4–60: eventGraphicPen Parameters (3)

Key	Type	Bounds	Options
keyStrokeLength ('StrL')	typeInteger	1..15	flagsSingleParameter
keyLightDark ('LgDr')	typeInteger	0..100	flagsSingleParameter
keyStrokeDirection ('SDir')	typeStrokeDirection		flagsEnumeratedParameter

eventHalftoneScreen ('HlfS')

Image Reference

Table 4–61: eventHalftoneScreen Parameters (3)

Key	Type	Bounds	Options
keyHalftoneSize ('HlSz')	typeInteger	1..12	flagsSingleParameter
keyContrast ('Cntr')	typeInteger	0..50	flagsSingleParameter
keyScreenType ('ScrT')	typeScreenType		flagsEnumeratedParameter

eventInkOutlines ('InkO')

Image Reference

Table 4–62: eventInkOutlines Parameters (3)

Key	Type	Bounds	Options
keyStrokeLength ('StrL')	typeInteger	1..50	flagsSingleParameter
keyDarkIntensity ('Drkl')	typeInteger	0..50	flagsSingleParameter
keyLightIntensity ('Lghl')	typeInteger	0..50	flagsSingleParameter

eventMosaic_PLUGIN ('MscT')

Image Reference

Table 4–63: eventMosaic_PLUGIN Parameters (3)

Key	Type	Bounds	Options
keyTileSize ('TlSz')	typeInteger	2..100	flagsSingleParameter
keyGroutWidth ('GrtW')	typeInteger	1..15	flagsSingleParameter
keyLightenGrout ('LghG')	typeInteger	0..10	flagsSingleParameter

eventNeonGlow ('NGlw')

Image Reference

Table 4–64: eventNeonGlow Parameters (3)

Key	Type	Bounds	Options
keySize ('Sz ')	typeInteger	-24..24	flagsSingleParameter
keyBrightness ('Brgh')	typeInteger	0..50	flagsSingleParameter
keyColor ('Clr ')	typeColor		flagsSingleParameter

eventNotePaper ('NtPr')

Image Reference

Table 4–65: eventNotePaper Parameters (3)

Key	Type	Bounds	Options
keyImageBalance ('ImgB')	typeInteger	0..50	flagsSingleParameter
keyGraininess ('Grns')	typeInteger	0..20	flagsSingleParameter
keyRelief ('Rlf ')	typeInteger	0..25	flagsSingleParameter

eventOceanRipple ('OcnR')

Image Reference

Table 4–66: eventOceanRipple Parameters (2)

Key	Type	Bounds	Options
keyRippleSize ('RplS')	typeInteger	1..15	flagsSingleParameter
keyRippleMagnitude ('RplM')	typeInteger	0..20	flagsSingleParameter

eventPaintDaubs ('PntD')

Image Reference

Table 4–67: eventPaintDaubs Parameters (3)

Key	Type	Bounds	Options
keySize ('Sz ')	typeInteger	1..50	flagsSingleParameter
keySharpness ('Shrp')	typeInteger	0..40	flagsSingleParameter
keyBrushType ('BrsT')	typeInteger	0..25	flagsEnumeratedParameter

eventPaletteKnife ('PltK')

Image Reference

Table 4–68: eventPaletteKnife Parameters (3)

Key	Type	Bounds	Options
keyStrokeSize ('StrS')	typeInteger	1..50	flagsSingleParameter
keyStrokeDetail ('StDt')	typeInteger	1..3	flagsSingleParameter
keySoftness ('Sftn')	typeInteger	0..10	flagsEnumeratedParameter

eventPatchwork ('Ptch')

Image Reference

Table 4–69: eventPatchwork Parameters (2)

Key	Type	Bounds	Options
keySquareSize ('SqrS')	typeInteger	1..10	flagsSingleParameter
keyRelief ('Rlf ')	typeInteger	0..25	flagsSingleParameter

eventPhotocopy ('Phtc')

Image Reference

Table 4–70: eventPhotocopy Parameters (2)

Key	Type	Bounds	Options
keyDetail ('Dtl ')	typeInteger	1..24	flagsSingleParameter
keyDarkness ('Drkn')	typeInteger	1..50	flagsSingleParameter

eventPlaster ('Plst')

Image Reference

Table 4–71: eventPlaster Parameters (3)

Key	Type	Bounds	Options
keyImageBalance ('ImgB')	typeInteger	0..50	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..15	flagsSingleParameter
keyLightPosition ('LghP')	typeLightPosition		flagsEnumeratedParameter

eventPlasticWrap ('PlsW')

Image Reference

Table 4–72: eventPlasticWrap Parameters (3)

Key	Type	Bounds	Options
keyHighlightStrength ('HghS')	typeInteger		flagsSingleParameter
keyDetail ('Dtl ')	typeInteger		flagsSingleParameter
keySmoothness ('Smth')	typeInteger		flagsSingleParameter

eventPosterEdges ('PstE')

Image Reference

Table 4–73: eventPosterEdges Parameters (3)

Key	Type	Bounds	Options
keyEdgeThickness ('EdgT')	typeInteger	0..10	flagsSingleParameter
keyEdgeIntensity ('Edgl')	typeInteger	0..10	flagsSingleParameter
keyPosterization ('Pstr')	typeInteger	0..6	flagsSingleParameter

eventReticulation ('Rtcl')

Image Reference

Table 4–74: eventReticulation Parameters (3)

Key	Type	Bounds	Options
keyDensity ('Dnst')	typeInteger	0..50	flagsSingleParameter
keyBlackLevel ('BlcL')	typeInteger	0..50	flagsSingleParameter
keyWhiteLevel ('WhtL')	typeInteger	0..50	flagsSingleParameter

eventRoughPastels ('RghP')

Image Reference

Table 4–75: eventRoughPastels Parameters (3)

Key	Type	Bounds	Options
keyStrokeLength ('StrL')	typeInteger	0..40	flagsSingleParameter
keyStrokeDetail ('StDt')	typeInteger	1..20	flagsSingleParameter
keyLightDirection ('LghD')	typeLightDirection		flagsEnumeratedParameter
keyTextureType ('TxtT')	typeTextureType		flagsOptionalSingleParameter
keyTextureType ('TxtT')	typeAlias		flagsOptionalSingleParameter
keyScaling ('Scln')	typeInteger	50..200	flagsSingleParameter
keyRelief ('Rlf ')	typeInteger	0..50	flagsSingleParameter
keyInvertTexture ('InvT')	typeBoolean		flagsSingleParameter

eventSmudgeStick ('SmdS')

Image Reference

Table 4–76: eventSmudgeStick Parameters (3)

Key	Type	Bounds	Options
keyStrokeLength ('StrL')	typeInteger	0..10	flagsSingleParameter
keyHighlightArea ('HghA')	typeInteger	0..20	flagsSingleParameter
keyIntensity ('Intn')	typeInteger	0..10	flagsSingleParameter

eventSpatter ('Spt ')

Image Reference

Table 4–77: eventSpatter Parameters (2)

Key	Type	Bounds	Options
keySprayRadius ('SprR')	typeInteger	0..25	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..15	flagsSingleParameter

eventSponge ('Spng')

Image Reference

Table 4–78: eventSponge Parameters (3)

Key	Type	Bounds	Options
keyBrushSize ('BrsS')	typeInteger	0..10	flagsSingleParameter
keyDefinition ('Dfnt')	typeInteger	0..25	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	0..15	flagsSingleParameter

eventSprayedStrokes ('SprS')

Image Reference

Table 4–79: eventSprayedStrokes Parameters (3)

Key	Type	Bounds	Options
keyStrokeLength ('StrL')	typeInteger	0..20	flagsSingleParameter
keySprayRadius ('SprR')	typeInteger	0..25	flagsSingleParameter
keyStrokeDirection ('SDir')	typeStrokeDirection		flagsEnumeratedParame-ter

eventStainedGlass ('StnG')

Image Reference

Table 4–80: eventStainedGlass Parameters (3)

Key	Type	Bounds	Options
keyCellSize ('ClSz')	typeInteger	2..50	flagsSingleParameter
keyBorderThickness ('BrdT')	typeInteger	1..20	flagsSingleParameter
keyLightIntensity ('Lghl')	typeInteger	0..10	flagsSingleParameter
keyLightDark ('LgDr')	typeInteger	0..100	flagsSingleParameter

eventStamp ('Stmp')

Image Reference

Table 4–81: eventStamp Parameters (2)

Key	Type	Bounds	Options
keyLightDark ('LgDr')	typeInteger	0..50	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..50	flagsSingleParameter

eventSumie ('Smie')

Image Reference

Table 4–82: eventSumie Parameters (3)

Key	Type	Bounds	Options
keyStrokeWidth ('StrW')	typeInteger	3..15	flagsSingleParameter
keyStrokePressure ('StrP')	typeInteger	0..15	flagsSingleParameter
keyContrast ('Cntr')	typeInteger	0..40	flagsSingleParameter

eventTexturizer ('Txtz')

Image Reference

Table 4–83: eventTexturizer Parameters (6)

Key	Type	Bounds	Options
keyLightDirection ('LghD')	typeLightDirection		flagsEnumeratedParameter
keyTextureType ('TxtT')	typeTextureType		flagsOptionalSingleParameter
keyTextureType ('TxtT')	typeAlias		flagsOptionalSingleParameter
keyScaling ('Scln')	typeInteger	50..200	flagsSingleParameter
keyRelief ('Rlf ')	typeInteger	0..50	flagsSingleParameter
keyInvertTexture ('InvT')	typeBoolean		flagsSingleParameter

eventTornEdges ('TrnE')

Image Reference

Table 4–84: eventTornEdges Parameters (3)

Key	Type	Bounds	Options
keyImageBalance ('ImgB')	typeInteger	0..50	flagsSingleParameter
keySmoothness ('Smth')	typeInteger	1..15	flagsSingleParameter
keyContrast ('Cntr')	typeInteger	1..25	flagsSingleParameter

eventUnderpainting ('Undr')

Image Reference

Table 4–85: eventUnderpainting Parameters (8)

Key	Type	Bounds	Options
keyBrushSize ('BrsS')	typeInteger	0..40	flagsSingleParameter
keyTextureCoverage ('TxtC')	typeInteger	0..40	flagsSingleParameter
keyLightDirection ('LghD')	typeLightDirection		flagsEnumeratedParameter
keyTextureType ('TxtT')	typeTextureType		flagsOptionalSingleParameter
keyTextureType ('TxtT')	typeAlias		flagsOptionalSingleParameter
keyScaling ('Scln')	typeInteger	50..200	flagsSingleParameter
keyRelief ('Rlf ')	typeInteger	0..50	flagsSingleParameter
keyInvertTexture ('InvT')	typeBoolean		flagsSingleParameter

eventWaterPaper ('WtrP')

Image Reference

Table 4–86: eventWaterPaper Parameters (3)

Key	Type	Bounds	Options
keyFiberLength ('FbrL')	typeInteger	3..50	flagsSingleParameter
keyBrightness ('Brgh')	typeInteger	0..100	flagsSingleParameter
keyContrast ('Cntr')	typeInteger	1..100	flagsSingleParameter

eventWatercolor ('Wtrc')

Image Reference

Table 4–87: eventWatercolor Parameters (3)

Key	Type	Bounds	Options
keyBrushDetail ('BrsD')	typeInteger	1..14	flagsSingleParameter
keyShadowIntensity ('Shdl')	typeInteger	0..10	flagsSingleParameter
keyTexture ('Txtr')	typeInteger	1..3	flagsSingleParameter

event3DTransform('TdT ')

Image Reference

Table 4–88: event3DTransform Parameters (5)

Key	Type	Bounds	Options
keyManipulationFOV ('Mfov')	typeFloat		flagsSingleParameter
keyConstructionFOV ('Cfov')	typeFloat		flagsSingleParameter
keyBackground ('Bckg')	typeBoolean		flagsSingleParameter
keyRenderFidelity ('Rfid')	typeAmount		flagsEnumeratedParameter
key3DAntiAlias ('Alis')	typeAntiAlias		flagsEnumeratedParameter

eventClouds ('Clds')

Image Reference

`takes no parameters`

eventDifferenceClouds ('DfrC')

Image Reference

`takes no parameters`

eventColorHalftone ('ClrH')

Image Reference

Table 4–89: eventColorHalftone Parameters (5)

Key	Type	Bounds	Options
keyRadius ('Rds ')	typeInteger	4 . . 127	flagsSingleParameter
keyAngle1 ('Ang1')	typeInteger	-360..360	flagsSingleParameter
keyAngle2 ('Ang2')	typeInteger	-360..360	flagsSingleParameter
keyAngle3 ('Ang3')	typeInteger	-360..360	flagsSingleParameter
keyAngle4 ('Ang4 ')	typeInteger	-360..360	flagsSingleParameter

eventCrystallize ('Crst')

Image Reference

Table 4–90: eventCrystallize Parameters (1)

Key	Type	Bounds	Options
keyCellSize ('ClSz')	typeInteger	3 . . 300	flagsSingleParameter

eventDeInterlace ('Dntr')

Image Reference

Table 4–91: eventDeInterlace Parameters (2)

Key	Type	Bounds	Options
keyInterlaceEliminate-Type ('IntE')	typeInterlaceEliminate-Type		flagsEnumeratedParame-ter
keyInterlaceCreateType ('IntC')	typeInterlaceCreateType		flagsEnumeratedParame-ter

eventExtrude ('Extr')

Image Reference

Table 4–92: eventExtrude Parameters (6)

Key	Type	Bounds	Options
keyExtrudeSize ('ExtS')	typeInteger	2 . . 255	flagsSingleParameter
keyExtrudeDepth ('ExtD')	typeInteger	1 . . 255	flagsSingleParameter
keyExtrudeSolidFace ('ExtF')	typeBoolean		flagsSingleParameter
keyExtrudeMaskIncom-plete ('ExtM')	typeBoolean		flagsSingleParameter
keyExtrudeType ('ExtT')	typeExtrudeType		flagsEnumeratedParame-ter
keyExtrudeRandom ('ExtR')	typeExtrudeRandom		flagsEnumeratedParame-ter

eventLensFlare ('LnsF')

Image Reference

Table 4–93: eventLensFlare Parameters (3)

Key	Type	Bounds	Options
keyBrightness ('Brgh')	typeInteger	10..300	flagsSingleParameter
keyFlareCenter ('FlrC')	classPoint		flagsSingleParameter
keyLens ('Lns ')	typeLens		flagsEnumeratedParame-ter

eventMezzotint ('Mztn')

Image Reference

Table 4–94: eventMezzotint Parameters (1)

Key	Type	Bounds	Options
keyMezzotintType ('MztT')	typeMezzotintType		flagsEnumeratedParame-ter

eventNTSCColors ('NTSC')

Image Reference
`takes no parameters`
targets: null

eventPointillize ('Pntl')

Image Reference}

Table 4–95: eventPointillize Parameters (1)

Key	Type	Bounds	Options
keyCellSize ('ClSz')	typeInteger	3 . . 300	flagsSingleParameter

eventRadialBlur ('RdlB')

Image Reference

Table 4–96: eventRadialBlur Parameters (3)

Key	Type	Bounds	Options
keyAmount ('Amnt')	typeIntegerl	1..100	flagsSingleParameter
keyBlurMethod ('BlrM')	typeBlurMethod		flagsEnumeratedParame-ter
keyBlurQuality ('BlrQ')	typeBlurQuality		flagsEnumeratedParame-ter

eventSmartBlur ('SmrB')

Image Reference

Table 4–97: eventSmartBlur Parameters (4)

Key	Type	Bounds	Options
keyRadius ('Rds ')	typeFloat/unitPixel	0.1..100	flagsSingleParameter
keyThreshold ('Thsh')	typeFloat	0.1..100	flagsEnumeratedParame-ter
keySmartBlurQuality ('SmBQ')	typeSmartBlurQuality		flagsEnumeratedParame-ter
keySmartBlurMode ('SmBM')	typeSmartBlurMode		flagsEnumeratedParame-ter

eventSolarize ('Slrz')

Image Reference
`takes no parameters`
targets: null

eventTiles ('Tls ')

Image Reference

Table 4–98: eventTiles Parameters (3)

Key	Type	Bounds	Options
keyTileNumber ('TlNm')	typeIntegerl	1..99	flagsSingleParameter
keyTileOffset ('TlOf')	typeIntegerl	1..99	flagsSingleParameter
keyFillColor ('FlCl')	typeFillColor		flagsEnumeratedParame-ter

eventVariations ('Vrtn')

Image Reference

Table 4–99: eventVariations Parameters (10)

Key	Type	Bounds	Options
keyRedGamma ('RdGm')	typeFloat		flagsSingleParameter
keyGreenGamma ('GrnG')	typeFloat		flagsSingleParameter
keyBlueGamma ('BlGm')	typeFloat		flagsSingleParameter
keyRedWhitePoint ('RdWh')	typeIntegerl		flagsSingleParameter
keyGreenWhitePoint ('GrnW')	typeIntegerl		flagsSingleParameter
keyBlueWhitePoint ('BlWh')	typeInteger		flagsSingleParameter
keyRedBlackPoint ('RdBl')	typeInteger		flagsSingleParameter
keyGreenBlackPoint ('GrnB')	typeInteger		flagsSingleParameter
keyBlueBlackPoint ('BlBl')	typeInteger		flagsSingleParameter
keySaturation ('Strt')	typeInteger		flagsSingleParameter

eventWind ('Wnd ')

Image Reference

Table 4–100: eventWind Parameters (2)

Key	Type	Bounds	Options
keyWindMethod ('WndM')	typeWindMethod		flagsEnumeratedParameter
keyDirection ('Drct')	typeDirection		flagsEnumeratedParameter

eventDisplace ('Dspl')

Image Reference

Table 4–101: eventDisplace Parameters (5)

Key	Type	Bounds	Options
keyHorizontalScale ('HrzS')	typeInteger		flagsSingleParameter
keyVerticalScale ('VrtS')	typeInteger		flagsSingleParameter
keyDisplacementMap ('DspM')	typeDisplacementMap		flagsEnumeratedParameter
keyUndefinedArea ('UndA')	typeUndefinedArea		flagsEnumeratedParameter
keyDisplaceFile ('DspF')	typeAlias		flagsSingleParameter

eventPinch ('Pnch')

Image Reference

Table 4–102: eventPinch Parameters (1)

Key	Type	Bounds	Options
keyAmount ('Amnt')	typeInteger	-100 . . 100	flagsSingleParameter

eventPolar Coordinates ('Plr ')

Image Reference

Table 4–103: eventPolar Parameters (1)

Key	Type	Bounds	Options
keyConvert ('Cnvr')	typeConvert		flagsEnumeratedParame-ter

eventRipple ('Rple')

Image Reference

Table 4–104: eventRipple Parameters (2)

Key	Type	Bounds	Options
keyAmount ('Amnt')	typeInteger	-999 . . 999	flagsSingleParameter
keyRippleSize ('RplS')	typeRippleSize		flagsEnumeratedParame-ter

eventShear ('Shr ')

Image Reference

Table 4–105: eventShear Parameters (4)

Key	Type	Bounds	Options
keyShearPoints ('ShrP')	classPoint		flagsListParameter
keyUndefinedArea ('UndA')	typeUndefinedArea		flagsEnumeratedParame-ter
keyShearSt ('ShrS')	typeInteger		flagsSingleParameter
keyShearEd ('ShrE')	typeInteger		flagsSingleParameter

eventSpherize ('Sphr')

Image Reference

Table 4–106: eventSpherize Parameters (2)

Key	Type	Bounds	Options
keyAmount ('Amnt')	typeInteger	-100 . . 100	flagsSingleParameter
keySpherizeMode ('SphM')	typeSpherizeMode		flagsEnumeratedParame-ter

eventTwirl ('Twrl')

Image Reference

Table 4–107: eventTwirl Parameters (1)

Key	Type	Bounds	Options
keyAngle ('Angl')	typeInteger	-999 . . 999	flagsSingleParameter

eventWave ('Wave')

Image Reference

Table 4–108: eventWave Parameters (10)

Key	Type	Bounds	Options
keyWaveType ('Wvtp')	typeWaveType		flagsEnumeratedParameter
keyNumberOfGenerators ('NmbG')	typeIntegerl		flagsSingleParameter
keyWavelengthMin ('WLMn')	typeIntegerl		flagsSingleParameter
keyWavelengthMax ('WLMx')	typeIntegerl		flagsSingleParameter
keyAmplitudeMin ('AmMn')	typeIntegerl		flagsSingleParameter
keyAmplitudeMax ('AmMx')	typeInteger		flagsSingleParameter
keyScaleHorizontal ('SclH')	typeInteger		flagsSingleParameter
keyScaleVertical ('SclV')	typeInteger		flagsSingleParameter
keyUndefinedArea ('UndA')	typeUndefinedArea		flagsEnumeratedParameter
keyRandomSeed ('RndS')	typeInteger		flagsSingleParameter

eventZigZag ('ZgZg')

Image Reference

Table 4–109: eventZigZag Parameters (3)

Key	Type	Bounds	Options
keyAmount ('Amnt')	typeInteger	-100 . . 100	flagsSingleParameter
keyNumberOfRipples ('NmbR')	typeInteger	1 . . 20	flagsSingleParameter
keyZigZagType ('ZZTy')	typeZigZagType		flagsEnumeratedParameter

eventLightingEffects ('LghE')

Image Reference

Table 4–110: eventLightingEffects Parameters (11)

Key	Type	Bounds	Options
keyGloss ('Glos')	typeIntegerl		flagsSingleParameter
keyMaterial ('Mtrl')	typeIntegerl		flagsSingleParameter
keyExposure ('Exps')	typeIntegerl		flagsSingleParameter
keyAmbientBrightness ('AmbB')	typeIntegerl		flagsSingleParameter
keyAmbientColor ('AmbC')	classRGBColor		flagsSingleParameter
keyBumpChannel ('BmpC')	#ChR		flagsSingleParameter
keyWhiteIsHigh ('WhHi')	typeBoolean		flagsSingleParameter
keyBumpAmplitude ('BmpA')	typeInteger		flagsSingleParameter
keyFrameWidth ('FrmW')	typeFloat		flagsSingleParameter
keyCurrentLight ('CrnL')	typeInteger		flagsSingleParameter
keyLightSource ('LghS')	classLightSource		flagsListParameter

eventTextureFill ('TxtF')

Image Reference

Table 4–111: eventTextureFill Parameters (1)

Key	Type	Bounds	Options
keyTextureFile ('TxtF')	typeAlias		flagsSingleParameter

es

This chapter describes the events that can automate the tools in the Tools Palette. Scripting the tool palette is very similar to the other scripting commands, and scripting things like the Marquee Tool simply require making calls to pre-existing functions with different parameters.

This chapter contains a mix of objects and functions that allow you to script the tools in the tools palette.

Built-In Tool Classes

Marquee Tool Selection Classes

The following classes are to be used to modify the selection mask. They are used in the functions `Set`, `Intersect`, `Add` and `Subtract`.

classRectangle ('Rctn')

inherits from: none
contains: undefined

Table 5–112: classRectangle Members (4)

Key	Type	Bounds	Options
keyTop ('Top ')	typeFloat/unitPixels		flagsEnumeratedProperty
keyLeft ('Left')	typeFloat/unitPixels		flagsEnumeratedProperty
keyBottom ('Btom')	typeFloat/unitPixels		flagsEnumeratedProperty
keyRight ('Rght')	typeFloat/unitPixels		flagsEnumeratedProperty

classEllipse ('Elps')

inherits from: none
contains: undefined

Table 5–113: classEllipse Members (4)

Key	Type	Bounds	Options
keyTop ('Top ')	typeFloat/unitPixels		flagsEnumeratedProperty
keyLeft ('Left')	typeFloat/unitPixels		flagsEnumeratedProperty
keyBottom ('Btom')	typeFloat/unitPixels		flagsEnumeratedProperty
keyRight ('Rght')	typeFloat/unitPixels		flagsEnumeratedProperty

classSingleRow ('Sngr')

inherits from: none
contains: undefined

Table 5–114: classSingleRow Member (1)

Key	Type	Bounds	Options
keyTop ('Top ')	typeFloat/unitPixels		flagsEnumeratedProperty

classSingleColumn ('Sngc')

inherits from: none
contains: undefined

Table 5–115: classSingleColumn Member (1)

Key	Type	Bounds	Options
keyLeft ('Left'))	typeFloat/unitPixels		flagsEnumeratedProperty

classTool ('Tool')

inherits from: classElement
contains: undefined

Table 9–116: classTool Parameter (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty

classBrush ('Brsh')

inherits from: classElement
contains: undefined

Table 9–117: classBrush Parameter (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty

More Tools

classPencilTool ('PcTl')

inherits from: classTool
contains: undefined

Table 5–118: classPencilTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classPaintbrushTool ('PbTl')

inherits from: classTool
contains: undefined

Table 5–119: classPaintbrushTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classAirbrushTool ('AbTl')

inherits from: classTool
contains: undefined

Table 5–120: classAirbrushTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classEraserTool ('ErTl')

inherits from: classTool
contains: undefined

Table 5–121: classEraserTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classCloneStampTool ('ClTl')

inherits from: classTool
contains: undefined

Table 5–122: classCloneStampTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classPatternStampTool ('PaTl')

inherits from: classTool
contains: undefined

Table 5–123: classPatternStampTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classSmudgeTool ('SmTl')

inherits from: classTool
contains: undefined

Table 5–124: classSmudgeTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classBlurTool ('BlTl')

inherits from: classTool
contains: undefined

Table 5–125: classBlurTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classSharpenTool ('ShTl')

inherits from: classTool
contains: undefined

Table 5–126: classSharpenTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classsDodgeTool ('DdTl')

inherits from: classTool
contains: undefined

Table 5–127: classDodgeTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classBurnInTool ('BrTl')

inherits from: classTool
contains: undefined

Table 5–128: classBurnInTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classSaturationTool ('SrTl')

inherits from: classTool
contains: undefined

Table 5–129: classSaturationTool Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classPath('BrTl')

inherits from: classElement
contains: undefined

Table 5–130: classPath Member (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')			flagsSingleProperty

classClippingPath ('ClpP')

inherits from: none
contains: undefined

Table 5–131: classClippingPath Member (1)

Key	Type	Bounds	Options
keyPath ('Pth ')	typePathReference		flagsEnumeratedProperty
keyFlatness ('Fltn')	typeFloat		flagsSingleProperty

8. Element Action Events

This chapter describes of all the element events that are included with Photoshop.

Element Events are events that are related to containers of pixels. This includes layers, channels, selections, and masks.

Built-in Element Events

eventAdd ('Add ')

Image Reference

Table 6–132: eventAdd Parameters (2)

Key	Type	Bounds	Options
keyTo ('To ')	typeWildCard		flagsSingleParameter
keyInvert ('Invr')	typeBoolean		flagsOptionalSingleParameter

eventCopy ('copy')

Image Reference

takes no parameters
targets: null

eventCopyMerged ('CpyM')

Image Reference

takes no parameters
targets: null

eventCopyToLayer ('CpTL')

Image Reference

takes no parameters
targets: null

eventCut ('cut ')

Image Reference

Table 6–133: eventCut Parameter (1)

Key	Type	Bounds	Options
keyTo ('To ')	typeLocationReference ('#Lct')		flagsSingleParameter

eventCutToLayer ('CtTL')

Image Reference

takes no parameters
targets: null

eventDefinePattern ('DfnP')

Image Reference

takes no parameters
targets: null

eventDelete ('Dlt ')

Image Reference

Table 6–134: eventDelete Parameter (1)

Key	Type	Bounds	Options
keyApply ('Aply')	typeBoolean		flagsOptionalSingleParameter

eventDuplicate ('Dplc')

Image Reference

Table 6–135: eventDuplicate Parameters (5)

Key	Type	Bounds	Options
keyTo ('To ')	typeElementReference ('#ElR')		flagsOptionalSingleParameter
keyName ('Nm ')	typeText		flagsOptionalSingleParameter
keyUntitled ('Untl')	typeBoolean		flagsOptionalSingleParameter
keyMerged ('Mrgd')	typeBoolean		flagsOptionalSingleParameter
keyInvert ('Invr')	typeBoolean		flagsOptionalSingleParameter

eventExchange ('Exch')

Image Reference

takes no parameters
targets: null

eventFlip ('Flip')

Image Reference

Table 6–136: eventFlip Parameters (2)

Key	Type	Bounds	Options
keyAxis ('Axis')	typeOrientation ('Ornt')		flagsEnumeratedParameter
keyCopy ('Cpy ')	typeBoolean		flagsOptionalSingleParameter

eventGroup ('GrpL')

Image Reference

takes no parameters
targets: null

eventIntersect ('Intr')

Image Reference

Table 6–137: eventIntersect Parameters (2)

Key	Type	Bounds	Options
keyWith ('Wth')	typeWildCard		flagsSingleParameter
keyInvert ('Invr')	typeBoolean		flagsOptionalSingleParameter

eventMake ('Mk ')

Image Reference

Table 6–138: eventMake Parameters (11)

Key	Type	Bounds	Options
keyNew ('Nw ')	classElement ('Elmn')		flagsSingleParameter
keyAt ('At ')	typeElementReference ('#ElR')		flagsOptionalSingleParameter
keyUsing ('Usng')	typeWildCard		flagsOptionalSingleParameter
keyName ('Nm ')	typeText		flagsOptionalSingleParameter
keyChannelName ('ChnN')	typeText		flagsOptionalSingleParamete
keyLayerName ('LyrN')	typeText		flagsOptionalSingleParameter
keyCopy ('Cpy ')	typeBoolean		flagsOptionalSingleParameter
keyInvert ('Invr ')	typeBoolean		flagsOptionalSingleParameter
keyFrom ('From ')	typeElementReference ('#ElR ')		flagsOptionalSingleParameter
keyTolerance ('Tlrn ')	typeFloat		flagsOptionalSingleParamete
keyPosition ('Pstn ')	classPoint ('Pnt ')		flagsOptionalSingleParameter

eventMergeLayers ('MrgL')

Image Reference

Table 6–139: eventMergeLayers Parameter (1)

Key	Type	Bounds	Options
keyDuplicate ('Dplc')	typeBoolean		flagsOptionalSingleParameter

eventMergeVisible ('MrgV')

Image Reference

Table 6–140: eventMergeVisible Parameter (1)

Key	Type	Bounds	Options
keyDuplicate ('Dplc')	typeBoolean		flagsOptionalSingleParameter

eventMove ('move')

Image Reference

Table 6–141: eventMove Parameter (1)

Key	Type	Bounds	Options
keyTo ('To ')	typeLocationReference ('#Lct')		flagsSingleParameter

eventPaste ('past')

Image Reference

Table 6–142: eventPaste Parameters (2)

Key	Type	Bounds	Options
keyAs ('As ')	typeClass		flagsOptionalSingleParameter
keyAntiAlias ('AntA')	typeBoolean		flagsOptionalSingleParameter

eventPasteInto ('PstI')

Image Reference

Table 6–143: eventPasteInto Parameter (1)

Key	Type	Bounds	Options
keyAntiAlias ('AntA')	typeBoolean		flagsOptionalSingleParameter

eventPasteOutside ('PstO')

Image Reference

Table 6–144: eventPasteOutside Parameter (1)

Key	Type	Bounds	Options
keyAntiAlias ('AntA')	typeBoolean		flagsOptionalSingleParameter

eventPlay ('Ply ')

Image Reference

Table 6–145: eventPlay Parameter (1)

Key	Type	Bounds	Options
keyContinue ('Cntn')	typeBoolean		flagsOptionalSingleParameter

eventRemoveBlackMatte ('RmvB')

Image Reference

takes no parameters

eventRemoveLayerMask ('RmvL')

Image Reference

takes no parameters

eventRemoveWhiteMatte ('RmvW')

Image Reference

takes no parameters

eventRotate ('Rtte')

Image Reference

Table 6–146: eventRotate Parameters (2)

Key	Type	Bounds	Options
keyAngle ('Angl')	unitAngle ('#Ang')		flagsEnumeratedParameter
keyCopy ('Cpy ')	typeBoolean		flagsOptionalSingleParameter

eventSelect ('slct')

Image Reference

Table 6–147: eventSelect Parameters (3)

Key	Type	Bounds	Options
keyUsing ('Usng')	typeFSS		flagsOptionalSingleParameter
keyColorCorrection ('ClrC')	classColorCorrection ('ClrC')		flagsListParameter
keyMethod ('Mthd')	typeCorrectionMethod ('CrcM')		flagsEnumeratedParameter

eventSet ('setd')

Image Reference

Table 6–148: eventSet Parameters (6)

Key	Type	Bounds	Options
key_Source ('Srce')	typeWildCard		flagsSingleParameter
keyInvert ('Invr')	typeBoolean		flagsOptionalSingleParameter
keyFeather ('Fthr')	unitPixels ('#Pxl')		flagsOptionalEnumeratedParameter

Table 6–148: eventSet Parameters (6)

Key	Type	Bounds	Options
keyTolerance ('Tlrn')	unitPixels ('#Pxl')		flagsOptionalEnumerated-Parameter
keyMerged ('Mrgd')	typeBoolean		flagsOptionalSingleParam-eter
keyAntiAlias ('AntA')	typeBoolean		flagsOptionalSingleParam-eter

eventSplitChannels ('SplC')

Image Reference

takes no parameters

eventStop ('Stop')

Image Reference

Table 6–149: eventStop Parameters (2)

Key	Type	Bounds	Options
keyMessage ('Msge')	typeText		flagsSingleParameter
keyContinue ('Cntn')	typeBoolean		flagsOptionalSingleParam-eter

eventSubtract ('Sbtr')

Image Reference

Table 6–150: eventSubtract Parameters (2)

Key	Type	Bounds	Options
keyFrom ('Frm ')	typeWildCard		flagsSingleParameter
keyInvert ('Invr')	typeBoolean		flagsOptionalSingleParam-eter

eventTransform ('Trnf')

Image Reference

Table 6–151: eventTransform Parameters (15)

Key	Type	Bounds	Options
keyPosition ('Pstn')	classPoint ('Pnt ')		flagsOptionalSingleParam-eter
keyRelative ('Rltv')	typeBoolean		flagsOptionalSingleParam-eter
keyWidth ('Wdth')	unitDistance ('#Rlt')		flagsOptionalEnumerated-Parameter
keyHeight ('Hght')	unitDistance ('#Rlt')		flagsOptionalEnumerated-Parameter
keyConstrainProportions ('CnsP')	typeBoolean		flagsOptionalSingleParam-eter
keySkew ('Skew')	classPoint ('Pnt ')		flagsOptionalSingleParam-eter

Table 6–151: eventTransform Parameters (15)

Key	Type	Bounds	Options
keyAngle ('Angl')	unitAngle ('#Ang')		flagsOptionalEnumerated-Parameter
keyOffset ('Ofst')	classOffset ('Ofst')		flagsOptionalSingleParam-eter
keyFreeTransformCenter-State ('FTcs')	typeQuadCenterState ('QCSt')		flagsOptionalEnumerated-Parameter
keyPerspectiveIndex ('Prsp')	keyPerspectiveIndex ('Prsp')		flagsOptionalSingleParam-eter
keyUsing ('Usng')	classPoint ('Pnt ')		flagsOptionalSingleParam-eter
keyCrossover ('Crss')	classPoint ('Pnt ')		flagsOptionalSingleParam-eter
keyTwist ('Twst')	keyTwist ('Twst')		flagsOptionalSingleParam-eter
keyCopy ('Cpy ')	keyCopy ('Cpy ')		flagsOptionalSingleParam-eter
keyLastTransform ('LstT')	typeBoolean		flagsOptionalSingleParam-eter

eventUndo ('undo')

Image Reference

takes no parameters

eventUngroup ('Ungr')

Image Reference

takes no parameters

9. Document Events

This chapter describes all the document events that are included with Photoshop.

Document Events are events related to documents and containers of elements. Many of these events can be found in the FIle and Edit menus.

Built-In Document Events

eventCanvasSize ('CnvS')

Image Reference

Table 7–152: eventCanvasSize Parameters (4)

Key	Type	Bounds	Options
keyWidth ('Wdth')	unitDistance ('#Rlt')		flagsOptionalEnumerated-Parameter
keyHeight ('Hght')	unitDistance ('#Rlt')		flagsOptionalEnumerated-Parameter
keyHorizontal ('Hrzn')	typeHorizontalLocation ('HrzI')		flagsEnumeratedParame-ter
keyVertical ('Vrtc')	typeVerticalLocation ('VrtI')		flagsEnumeratedParame-ter

eventClose ('Cls ')

Image Reference

Table 7–153: eventClose Parameter (1)

Key	Type	Bounds	Options
keySaving ('Svng')	typeYesNo ('Ysn ')		flagsEnumeratedParame-ter

eventConvertMode ('CnvM')

Image Reference

Table 7–154: eventConvertMode Parameters (3)

Key	Type	Bounds	Options
keyTo ('To ')	typeTypeClassModeOr-ClassMode ('#TyM')		flagsOptionalEnumerated-Parameter
keyFlatten ('Fltn')	typeBoolean		flagsOptionalEnumerated-Parameter
keyInterpolation ('Intr')	typeInterpolation ('Intp')		flagsOptionalEnumerated-Parameter

eventCrop ('Crop')

Image Reference

Table 7–155: eventCrop Parameters (5)

Key	Type	Bounds	Options
keyTo ('T ')	classRectangle ('Rxtn')		flagsOptionalSingleParam-eter
keyAngle ('Angl')	unitAngle ('#Ang')		flagsOptionalEnumerated-Parameter

Table 7–155: eventCrop Parameters (5)

Key	Type	Bounds	Options
keyWidth ('Wdth')	unitPixels ('#Pxl')		flagsOptionalEnumerated-Parameter
keyHeight ('Hght')	unitPixels ('#Pxl')		flagsOptionalEnumerated-Parameter
keyResolution ('Rslt')	unitPixels ('#Pxl')		flagsOptionalEnumerated-Parameter

eventExport ('Expr')

Image Reference

Table 7–156: eventExport Parameter (1)

Key	Type	Bounds	Options
keyUsing ('Usng')	typeClassTextExport ('#CTE')		flagsSingleParameter

eventFlattenImage ('FltI')

Image Reference

takes no parameters
targets: null

eventBlur ('Blr ')

Image Reference

takes no parameters
targets: null

eventImageSize ('ImgS')

Image Reference

Table 7–157: eventImageSize Parameters (5)

Key	Type	Bounds	Options
keyWidth('Wdth')	unitDistance ('#Rlt')		flagsOptionalEnumerated-Parameter
keyHeight('Hght')	unitDistance ('#Rlt')		flagsOptionalEnumerated-Parameter
keyResolution('Rslt')	unitDensity ('#Rsl')		flagsOptionalEnumerated-Parameter
keyConstrainProportions ('CnsP')	typeBoolean		flagsOptionalSingleParam-eter
keyInterpolation ('Intr')	typeInterpolation ('Intp')		flagsOptionalEnumerated-Parameter

eventPrint ('Prnt')

Image Reference

takes no parameters
targets: null

eventQuit ('quit')

Image Reference

takes no parameters
targets: null

eventRevert ('Rvrt')

Image Reference
takes no parameters
targets: null

eventSave ('save')

Image Reference

Table 7–158: eventSave Parameters (9)

Key	Type	Bounds	Options
keyAs ('As ')	typeClassStringFormat ('#CIS')		flagsEnumeratedParame-ter
keyIn ('In ')	typeFSS		flagsSingleParameter
keyCopy ('Cpy ')	typeBoolean		flagsSingleParameter
keyExtension ('Extn')	typeBoolean		flagsSingleParameter
keyLowerCase ('LwCs')	typeBoolean		flagsSingleParameter
keyFlatten ('Fltt')	typeBoolean		flagsSingleParamete
keyAlphaChannels ('AlpC')	typeBoolean		flagsSingleParameter
keyNonImageData ('NnIm')	typeBoolean		flagsSingleParameter
keyPreview ('Prvw')	typePreview ('Prvw')		flagsEnumeratedListPa-rameter

eventTakeMergedSnapshot ('TkMr')

Image Reference

takes no parameters
targets: null

eventTakeSnapshot ('TkSn')

Image Reference

takes no parameters
targets: null

10. File Action Events

This chapter describes of all the file events that are included with Photoshop.

File Events are events related to files and containers of documents. Most of these events are in the File menu.

Built-in File Events

eventBatch ('Btch')

Image Reference

Table 8–159: eventBatch Parameters (8)

Key	Type	Bounds	Options
keyOverrideOpen ('OvrO')	typeBoolean		flagsOptionalSingleParameter
keyFolders ('Fldr')	typeBoolean		flagsOptionalSingleParameter
keyUsing ('Usng')	typeActionReference ('#Act')		flagsEnumeratedParameter
keyTo ('T ')	typeFSS		flagsOptionalSingleParameter
keyOverrideSave ('Ovrd')	typeBoolean		flagsOptionalSingleParameter
keySaveAndClose ('SvAn')	typeBoolean		flagsOptionalSingleParameter
keyLog ('Log ')	typeFSS		flagsOptionalSingleParameter
keyName ('Nm ')	typeText		flagsOptionalSingleParameter

eventImport ('Impr')

Image Reference

Table 8–160: eventImport Parameter (1)

Key	Type	Bounds	Options
keyUsing ('Usng')	typeClassTextImport ('#ClT')		flagsSingleParameter

eventOpen ('Opn ')

Image Reference

Table 8–161: eventOpen Parameter (1)

Key	Type	Bounds	Options
keyAs ('As ')	typeStringClassFormat ('#StC')		flagsEnumeratedParameter

eventRasterize ('Rstr')

Image Reference

Table 8–162: eventRasterize Parameter (1)

Key	Type	Bounds	Options
keyAS ('As ')	typeFSS		flagsEnumeratedParameter

11. Classes and Formats

This chapter documents the different objects and classes you will encounter as you build your descriptors, and the appropriate parameters for each.

The format of the following is very similar to the previous chapters, but with some minor adjustments.

The `inherits from` tag indicates the class the current object or class is derived from, if any. The `contains` tag indicates the class is a container that may hold specific objects.

For instance, `document` can contain an object of type `channel`. Objects can contain multiple numbers of the same object, or can contain multiple numbers of different objects. In some instances, what the object contains can be either optional or required. For instance, a document can contain an adjustment layer, but it is not required to. Alternatively, all documents must have at least one channel associated with them.

Classes

classColorCorrection ('ClrC')

inherits from: none
contains: undefined

Table 9–163: classColorCorrection Parameters (4)

Key	Type	Bounds	Options
keyColors ('Clrs')	typeColors ('Clrs')		flagsEnumeratedProperty
keyCyan ('Cyn ')	typeInteger		flagsSingleProperty
keyMagenta ('Mgnt')	typeInteger		flagsSingleProperty
keyYellow ('Ylw ')	typeInteger		flagsSingleProperty
keyBlack ('Blck')	typeInteger		flagsSingleProperty

classColor ('Clr ')

inherits from: none
contains: undefined

Table 9–164: classColor Parameter (1)

Key	Type	Bounds	Options
keyColor ('Clr ')	typeClassColor ('#Clr')		flagsEnumeratedProperty

classPoint ('Pnt ')

inherits from: none
contains: undefined

Table 9–165: classPoint Parameter (2)

Key	Type	Bounds	Options
keyHorizontal ('Hrzn')	typeFloat		flagsSingleProperty
keyVertical ('Vrtc')	typeFloat		flagsSingleProperty

classOffset ('Ofst')

inherits from: classPoint
contains: undefined

Table 9–166: classOffset Parameter (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classPoint		flagsSingleProperty

classRGBColor ('RGBC')

inherits from: classcolor
contains: undefined

Table 9–167: classRGBColor Parameter (3)

Key	Type	Bounds	Options
keyRed ('Rd ')	typeFloat		flagsSingleProperty
keyGreen ('Grn ')	typeFloat		flagsSingleProperty
keyBlue ('Bl ')	typeFloat		flagsSingleProperty

classUnspecifiedColor ('UnsC')

inherits from: classColor
contains: undefined

Table 9–168: classUnspecifiedColor Parameter (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classColor		flagsSingleProperty

classGrayscale ('Grys')

inherits from: classColor
contains: undefined

Table 9–169: classUnspecifiedColor Parameter (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classColor		flagsSingleProperty
keyGray ('Gry ')	typeFloat		flagsSingleProperty

classCMYKColor ('CMYC')

inherits from: classColor
contains: undefined

Table 9–170: classCMYKColor Parameters (5)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classColor		flagsSingleProperty
keyCyan ('Cyn ')	typeFloat		flagsSingleProperty
keyMagenta ('Mgnt')	typeFloat		flagsSingleProperty
keyYellow ('Ylw ')	typeFloat		flagsSingleProperty
keyBlack ('Blck')	typeFloat		flagsSingleProperty

classHSBColor ('HSBC')

inherits from: classColor
contains: undefined

Table 9–171: classHSBColor Parameters (4)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classColor		flagsSingleProperty
keyHue ('H ')	typeFloat/unitAngle		flagsSingleProperty
keySaturation ('Strt')	typeFloat		flagsSingleProperty
keyBrightness ('Brgh')	typeFloat		flagsSingleProperty

classLabColor ('LabC')

inherits from: classColor
contains: undefined

Table 9–172: classLabColor Parameters (4)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classColor		flagsSingleProperty
keyLuminance ('Lmn ')	typeFloat		flagsSingleProperty
keyA ('A ')	typeFloat		flagsSingleProperty
keyB ('B ')	typeFloat		flagsSingleProperty

classBookColor ('BkCl')

inherits from: classColor
contains: undefined

Table 9–173: classBookColor Parameters (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classColor		flagsSingleProperty
keyBook ('Bk ')	typeText		flagsSingleProperty
keyName ('Nm ')	typeText		flagsSingleProperty

classFileInfo ('FlIn')

inherits from: none
contains: undefined

Table 9–174: classFileInfo Parameters (21)

Key	Type	Bounds	Options
keyCaption ('Cptn')	typeText		flagsSingleProperty
keyCaptionWriter ('CptW')	typeText		flagsSingleProperty
keyHeadline ('Hdln')	typeText		flagsSingleProperty
keySpecialInstructions ('Spcl')	typeText		flagsSingleProperty
keyKeywords ('Kywd')	typeText		flagsListProperty

Table 9–174: classFileInfo Parameters (21)

Key	Type	Bounds	Options
keyCategory ('Ctgr')	typeText		flagsSingleProperty
keySupplementalCatego-ries (SplC')	typeText		flagsListProperty
keyUrgency ('Urgn')	typeUrgency		flagsSingleProperty
keyByline ('Byln')	typeText		flagsSingleProperty
keyBylineTitle ('BylT')	typeText		flagsSingleProperty
keyCredit ('Crdt')	typeText		flagsSingleProperty
keySource ('Srce')	typeText		flagsSingleProperty
keyObjectName ('ObjN')	typeText		flagsSingleProperty
keyDateCreated ('DtCr')	typeText		flagsSingleProperty
keyCity ('City')	typeText		flagsSingleProperty
keyProvinceState ('PrvS')	typeText		flagsSingleProperty
keyCountryName ('CntN')	typeText		flagsSingleProperty
keyOrginalTransmission-Reference ('OrgT')	typeText		flagsSingleProperty
keyCopyright ('Cpyr')	typeText		flagsSingleProperty
keyCopyrightNotice ('CprN')	typeText		flagsSingleProperty
keyURL ('URL ')	typeText		flagsSingleProperty

classElement ('Elmn')

inherits from: none
contains: undefined
this class has no properties

classTool ('Tool')

inherits from: classElement
contains: undefined

Table 9–175: classTool Parameter (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty

classBrush ('Brsh')

inherits from: classElement
contains: undefined

Table 9–176: classBrush Parameter (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty

classDocument ('Dcmn')

inherits from: element
contains: undefined

Table 9–177: classDocument Parameters (18)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyName ('Nm ')	typeText		flagsSingleProperty
keyMode ('Md ')	typeClassMode		flagsEnumeratedProperty
keyWidth ('Wdth')	unitFloat/unitDistance		flagsEnumeratedProperty
keyHeight ('Hght')	unitFloat/unitDistance		flagsEnumeratedProperty
keyResolution ('Rslt')	unitFloat/unitDensity		flagsEnumeratedProperty
keyFill ('Fl ')	unitFloat/unitDensity		flagsEnumeratedProperty
keyWorkPath('WrPt')	classPath		flagsSingleProperty
keyCurrentHistoryState ('CrnH')	classHistoryState		flagsSingleProperty
keyHistoryBrushSource ('HstB')	classhistoryState		flagsSingleProperty
keyBackground ('Bckg')	classLayer		flagsSingleProperty
keySelection ('fsel')	classChannel		flagsSingleProperty
keyFileInfo ('FlIn')	classFileInfo		flagsSingleProperty
keyQuickMask ('QucM')	typeBoolean		flagsSingleProperty
keyGuides('Gdes')	classGuide		flagsListProperty
keyHistoryStates ('HsSt')	classHistoryState		flagsListProperty
keyLayerFXVisible ('lfxv')	classLayerFXVisible		flagsSingleProperty
keyGlobalAngle ('gblA')	classGlobalAngle		flagsSingleProperty

classPixel ('Pxl ')

inherits from: classElement
contains: undefined

Table 9–178: classPixel (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty

classLayer ('Lyr ')

inherits from: classElement
contains: undefined

Table 9–179: classLayer Parameters (11)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyName ('Nm ')	typeText		flagsSingleProperty
keyOpacity ('Opct')	unitFloat/unitPercent		flagsSingleProperty
keyMode ('Md ')	typeBlendMode		flagsEnumeratedProperty

Table 9–179: classLayer Parameters (11)

Key	Type	Bounds	Options
keyGroup ('Grup')	typeBoolean		flagsSingleProperty
keyFillNeutral ('FlNt')	typeBoolean		flagsSingleProperty
keyPreserveTransparency ('PrsT')	typeBoolean		flagsSingleProperty
keyUserMaskEnabled ('UsrM')	typeBoolean		flagsSingleProperty
keyUserMaskLinked ('Usrs')	typeBoolean		flagsSingleProperty
keyBlendRange ('Blnd')	classBlendRange		flagsListProperty
keyLayerEffects ('Lefx')	classLayerEffects		flagsSingleProperty

classBackgroundLayer ('BckL')

inherits from: classLayer
contains: undefined

Table 9–180: classBackgroundLayer (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classLayer		flagsSingleProperty

classAdjustmentLayer ('AdjL')

inherits from: classLayer
contains: undefined

Table 9–181: classAdjustmentLayer (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classLayer		flagsSingleProperty
keyType ('Type')	classAdjustment		flagsSingleProperty

classAdjustment ('Adjs')

inherits from: none
this class has no properties
contains: undefined

classTextLayer ('TxLy')

inherits from: classLayer
contains: undefined

Table 9–182: classTextLayer Parameters (11)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classLayer		flagsSingleProperty
keyText ('Txt ')	typeChar		flagsSingleProperty
keyTextClickPoint ('TxtC')	classPoint		flagsSingleProperty
keyTextData ('TxtD')	typeRawData		flagsSingleProperty

Table 9–182: classTextLayer Parameters (11)

Key	Type	Bounds	Options
keyFontName ('FntN')	typeChar		flagsSingleProperty
keyScript (Scrp')	typeInteger		flagsSingleProperty
keyAlignment ('Algn')	typeAlignment		flagsEnumeratedProperty
keySize ('Sz ')	typeFloat		flagsSingleProperty
keyLeading ('Ldng')	typeFloat		flagsSingleProperty
keyTracking ('Trck')	typeFloat		flagsSingleProperty
keyAntiAlias ('AntA')	typeBoolean		flagsSingleProperty
keyRotate ('Rtt ')	typeBoolean		flagsSingleProperty
keyOrientation ('Ornt')	typeOrientation		flagsSingleProperty
keyAutoKern ('AtKr')	typeBoolean		flagsEnumeratedProperty

classInvert ('Invr')

inherits from: classAdjustment
contains: none

Table 9–183: classInvert (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty

classLevels ('Lvls')

inherits from: classAdjustment
contains: undefined

Table 9–184: classLevels (4)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyUsing ('Usng')	typeFSS		flagsSingleProperty
keyAuto ('Auto')	typeBoolean		flagsSingleProperty
keyAdjustment ('Adjs')	classLevelsAdjustment		flagsListProperty

classCurves ('Crvs')

inherits from: classAdjustment
contains: undefined

Table 9–185: classCurves (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyUsing ('Usng')	typeFSS		flagsSingleProperty
keyAdjustment ('Adjs')	classLevelsAdjustment		flagsListProperty

classBrightnessContrast ('BrCn')

inherits from: classAdjustment
contains: undefined

Table 9–186: classBrightnessContrast (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyBrightness ('Brgh')	typeInteger		flagsSingleProperty
keyContrast ('Cntr')	typeInteger		flagsSingleProperty

classColorBalance ('ClrB')

inherits from: classAdjustment
contains: undefined

Table 9–187: classColorBalance (5)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyShadowLevels('ShdL')	typeInteger		flagsListProperty
keyMidtoneLevels ('MdtL')	typeInteger		flagsListProperty
keyHighlightLevels ('HghL')	typeInteger		flagsListProperty
keyPreserveLuminosity ('PrsL')	typeBoolean		flagsSingleProperty

classHueSaturation ('HStr')

inherits from: classAdjustment
contains: undefined

Table 9–188: classHueSaturation (4)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyColorize ('Clrz')	typeBoolean		flagsSingleProperty
keyUsing ('Usng')	typeFSS		flagsSingleProperty
keyAdjustment ('Adjs')	classClassHueSatHueSatV2		flagsListProperty

classSelectiveColor ('SlcC')

inherits from: classAdjustment
contains: undefined

Table 9–189: classSelectiveColor (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyColorCorrection ('ClrC')	classColorCorrection		flagsListProperty
keyMethod ('Mthd')	typeCorrectionMethod		flagsEnumeratedProperty

classChannelMixer ('ChnM')

inherits from: classAdjustment
contains: undefined

Table 9–190: classChannelMixer (9)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyRed ('Rd ')	classChannelMatrix		flagsSingleProperty
keyGreen ('Grn ')	classChannelMatrix		flagsSingleProperty
keyBlue ('Bl ')	classChannelMatrix		flagsSingleProperty
keyCyan ('Cyn ')	classChannelMatrix		flagsSingleProperty
keyMagenta ('Mgnt')	classChannelMatrix		flagsSingleProperty
keyYellow ('Ylw ')	classChannelMatrix		flagsSingleProperty
keyBlack ('Blck')	classChannelMatrix		flagsSingleProperty
keyMonochromatic ('Mnch')	typeBoolean		flagsSingleProperty

classThreshold ('Thrs')

inherits from: classAdjustment
contains: undefined

Table 9–191: classThreshold (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyLevel ('Lvl ')	typeInteger		flagsSingleProperty

classPosterize ('Pstr')

inherits from: classAdjustment
contains: undefined

Table 9–192: classPosterize (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classAdjustment		flagsSingleProperty
keyLevel ('Lvl ')	typeInteger		flagsSingleProperty

classChannel ('Chnl')

inherits from: classElement
contains: undefined

Table 9–193: classChannel (5)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyName ('Nm ')	typeText		flagsSingleProperty
keyColorIndicates ('Clrl')	typeMaskIndicator		flagsEnumeratedProperty
keyColor ('Clr ')	classColor		flagsSingleProperty
keyOpacity ('Opct')	typeInteger		flagsSingleProperty

classSpotColorChannel ('SCch')

inherits from: classElement
contains: undefined

Table 9–194: classSpotColorChannel (4)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyName ('Nm ')	typeText		flagsSingleProperty
keyColor ('Clr ')	classColor		flagsSingleProperty
keyOpacity ('Opct')	typeInteger		flagsSingleProperty

classMode ('Md ')

inherits from: none
contains: undefined

Table 9–195: classMode (1)

Key	Type	Bounds	Options
keyMode ('Md ')	typeClassMode		flagsEnumeratedProperty

classRGBColorMode ('RGBM')

inherits from: classMode
contains: undefined

Table 9–196: classRGBColorMode (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classMode		flagsSingleProperty

classBitmapMode ('BtmM')

inherits from: classMode
contains: undefined

Table 9–197: classBitmapMode (7)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classMode		flagsSingleProperty
keyResolution ('Rslt')	unitFloat/unitDensity		flagsEnumeratedProperty
keyMethod ('Mthd')	typeMethod		flagsEnumeratedProperty
keyFrequency ('Frqn')	unitFloat/unitDensity		flagsEnumeratedProperty
keyAngle ('Angl')	unitAngle		flagsEnumeratedProperty
keyShape ('Shp ')	typeShape		flagsEnumeratedProperty
keyHalftoneFile ('HlfF')	typeFSS		flagsSingleProperty

classGrayscaleMode ('Grys')

inherits from: classMode
contains: undefined

Table 9–198: classGrayscaleMode (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classMode		flagsSingleProperty
keyRatio ('Rt ')	typeInteger		flagsSingleProperty

classIndexedColorMode ('IndC')

inherits from: classMode
contains: undefined

Table 9–199: classIndexedColorMode (8)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classMode		flagsSingleProperty
keyPalette ('Plt ')	typeColorPalette		flagsEnumeratedProperty
keyPaletteFile ('PltF')	typeFSS		flagsSingleProperty
keyCustomPalette ('CstP')	classRGBColor		flagsListProperty
keyColors ('Clrs')	typeInteger		flagsSingleProperty
keyDither ('Dthr')	typeDither		flagsEnumeratedProperty
keyDitherQuality ('Dthq')	typeDitherQuality		flagsEnumeratedProperty
keyDitherPreserve ('Dthp')	typeBoolean		flagsSingleProperty

classDuotoneMode ('DtnM')

inherits from: classMode
contains: undefined

Table 9–200: classDuotoneMode (4)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classMode		flagsSingleProperty
keyUsing ('Usgn')	typeClassTextImport		flagsSingleProperty
keyInks ('Inks')	classDuotoneInk		flagsListProperty
keyOverprintColors ('OvrC')	classColor		flagsListProperty

classDuotoneInk ('DtnI')

inherits from: none
contains: undefined

Table 9–201: classDuotoneInk (4)

Key	Type	Bounds	Options
keyName ('Nm ')	typeText		flagsSingleProperty
keyColor ('Clr ')	classRGBColor		flagsSingleProperty
keyCurve ('Crv ')	classDuotonePoint		flagsListProperty
keyCurveFile ('CrvF')	typeFSS		flagsSingleProperty

classDuotonePoint ('DtnP')

inherits from: none
contains: undefined

Table 9–202: classDuotonePoint (2)

Key	Type	Bounds	Options
keyValue ('Vl ')	typeInteger		flagsSingleProperty
keyResponse ('Rspn')	typeFloat		flagsSingleProperty

classLabColorMode ('LbCl')

inherits from: classMode
contains: undefined

Table 9–203: classLabColorMode (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classMode		flagsSingleProperty

classCMYKColorMode ('CMYM')

inherits from: classMode
contains: undefined

Table 9–204: classCMYKColorMode (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classMode		flagsSingleProperty

classMultichannelMode ('MltC')

inherits from: classMode
contains: undefined

Table 9–205: classMultichannelMode (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classMode		flagsSingleProperty

classCommand ('Cmnd')

inherits from: classElement
contains: undefined

Table 9–206: classCommand (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty

classAction ('Actn')

inherits from: element
contains: command

Table 9–207: classAction (6)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyName ('Nm ')	typeChar		flagsSingleProperty
keyFunctionKey ('FncK')	typeInteger		flagsSingleProperty
keyShiftKey ('ShfK')	typeBoolean		flagsSingleProperty
keyCommandKey ('CmdK')	typeBoolean		flagsSingleProperty
keyColor ('Clr ')	typeColor		flagsEnumeratedProperty

classActionSet ('ASet')

inherits from: classElement
contains: undefined

Table 9–208: classActionSet (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyName ('Nm ')	typeChar		flagsSingleProperty

classApplication ('capp')

inherits from: none
contains: undefined

Table 9–209: classApplication (22)

Key	Type	Bounds	Options
keyCurrentToolOptions ('CrnT')	classProperty		flagsSingleProperty
keyAllToolOptions ('AlTl')	classProperty		flagsSingleProperty
keyBrushes ('Brsh')	classProperty		flagsSingleProperty
keyPreferences ('Prfr')	classProperty		flagsSingleProperty
keyColorTable ('ClrT')	classColor		flagsListProperty
keyColors ('Clrs')	classColor		flagsListProperty
keyForegroundColor ('FrgC')	classColor		flagsSingleProperty
keyBackgroundColor ('BckC')	classColor		flagsSingleProperty
keyWorkPath ('WrPt')	classPath		flagsSingleProperty
keyClippingPath ('ClPt')	classClippingPath		flagsSingleProperty
keySelection ('fsel')	classChannel		flagsSingleProperty
keyTargetPath ('Trgp')	classPath		flagsSingleProperty
keyCurrentHistoryState ('CrnH')	classHistoryState		flagsSingleProperty
keyHistoryBrushSource ('HstB')	classHistoryState		flagsSingleProperty
keyBackground ('Bckg')	classLayer		flagsSingleProperty
keyRGBSetup ('RGBS')	classProperty		flagsSingleProperty
keyCMYKSetup ('CMYS')	classProperty		flagsSingleProperty
keyGraySetup ('GrSt')	classProperty		flagsSingleProperty
keyProfileSetup ('PrfS')	classProperty		flagsSingleProperty
keyLayerEffects ('Lefx')	classLayerEffects		flagsSingleProperty
keyLayerFXVisible ('lfxv')	classLayerFXVisible		flagsSingleProperty
keyGlobalAngle ('gblA')	classGlobalAngle		flagsSingleProperty

classChannel ('Chnl')

inherits from: element

this class has no properties

contains: undefined

classLayerEffects ('Lefx')

inherits from: none
contains: undefined

Table 9–210: classLayerEffects (6)

Key	Type	Bounds	Options
keyScale ('Scl ')	unitPercent		flagsEnumeratedProperty
keyGlobalLightingAngle ('gagl')	unitAngle		flagsEnumeratedProperty
keyDropShadow ('DrSh')	classDropShadow		flagsSingleProperty
keyInnerShadow ('IrSh')	classInnerShadow		flagsSingleProperty
keyOuterGlow ('OrGl')	classOuterGlow		flagsSingleProperty
keyInnerGlow ('IrGl')	classInnerGlow		flagsSingleProperty
keyBevelEmboss ('ebbl')	classBevelEmboss		flagsSingleProperty

classDropShadow ('DrSh')

inherits from: none
contains: undefined

Table 9–211: classDropShadow (8)

Key	Type	Bounds	Options
keyMode ('Md ')	typeBlendMode		flagsEnumeratedProperty
keyColor ('Clr ')	classColor		flagsSingleProperty
keyOpacity ('Opct')	unitPercent		flagsEnumeratedProperty
keyUseGlobalAngle ('uglg')	typeBoolean		flagsSingleProperty
keyLocalLightingAngle ('lagl')	unitAngle		flagsEnumeratedProperty
keyDistance ('Dstn')	unitDistance		flagsEnumeratedProperty
keyBlur ('blur')	unitDistance		flagsEnumeratedProperty
keyIntensity ('Intn')	unitPercent		flagsEnumeratedProperty

classInnerShadow ('IrSh')

inherits from: none
contains: undefined

Table 9–212: classInnerShadow (8)

Key	Type	Bounds	Options
keyMode ('Md ')	typeBlendMode		flagsEnumeratedProperty
keyColor ('Clr ')	classColor		flagsSingleProperty
keyOpacity ('Opct')	unitPercent		flagsEnumeratedProperty
keyUseGlobalAngle ('uglg')	typeBoolean		flagsSingleProperty
keyLocalLightingAngle ('lagl')	unitAngle		flagsEnumeratedProperty
keyDistance ('Dstn')	unitDistance		flagsEnumeratedProperty
keyBlur ('blur')	unitDistance		flagsEnumeratedProperty
keyIntensity ('Intn')	unitPercent		flagsEnumeratedProperty

classOuterGlow ('OrGl')

inherits from: none
contains: undefined

Table 9–213: classOuterGlow (5)

Key	Type	Bounds	Options
keyMode ('Md ')	typeBlendMode		flagsEnumeratedProperty
keyColor ('Clr ')	classColor		flagsSingleProperty
keyOpacity ('Opct')	unitPercent		flagsEnumeratedProperty
keyBlur ('blur')	unitDistance		flagsEnumeratedProperty
keyIntensity ('Intn')	unitPercent		flagsEnumeratedProperty

classInnerGlow ('IrGl')

inherits from: none
contains: undefined

Table 9–214: classInnerGlow (6)

Key	Type	Bounds	Options
keyMode ('Md ')	typeBlendMode		flagsEnumeratedProperty
keyColor ('Clr ')	classColor		flagsSingleProperty
keyOpacity ('Opct')	unitPercent		flagsEnumeratedProperty
keyBlur ('blur')	unitDistance		flagsEnumeratedProperty
keyIntensity ('Intn')	unitPercent		flagsEnumeratedProperty
keyInnerGlowSource ('glwS')	typeInnerGlowSource		flagsEnumeratedProperty

classBevelEmboss ('ebbl')

inherits from: none
contains: undefined

Table 9–215: classBevelEmboss (13)

Key	Type	Bounds	Options
keyHighlightMode ('hglM')	typeBlendMode		flagsEnumeratedProperty
keyHighlightColor ('hglC')	classColor		flagsSingleProperty
keyHighlightOpacity ('hglO')	unitPercent		flagsEnumeratedProperty
keyShadowMode ('sdwM')	typeBlendMode		flagsEnumeratedProperty
keyShadowColor ('sdwC')	classColor		flagsSingleProperty
keyShadowOpacity ('sdwO')	unitPercent		flagsEnumeratedProperty
keyBevelStyle ('bvlS')	typeBevelEmbossStyle		flagsEnumeratedProperty
keyUseGlobalAngle ('uglg')	typeBoolean		flagsSingleProperty
keyLocalLightingAngle ('lagl')	unitAngle		flagsEnumeratedProperty
keyStrength ('srgh')	unitDistance		flagsEnumeratedProperty
keyBlur ('blur')	unitDistance		flagsEnumeratedProperty
keyBevelDirection ('bvlD')	typeBevelEmbossStamp-Style		flagsEnumeratedProperty
keyHistoryStateSource ('HsSS')	typeHistoryStateSource		flagsEnumeratedProperty

classLayerFXVisible ('lfxv')

inherits from: none
contains: undefined

Table 9–216: classLayerFXVisible (1)

Key	Type	Bounds	Options
keyLayerFXVisible ('lfxv')	typeBoolean		flagsSingleProperty

classGlobalAngle ('gblA')

inherits from: none
contains: undefined

Table 9–217: classGlobalAngle (1)

Key	Type	Bounds	Options
keyGlobalLightingAngle ('gagl')	unitAngle		flagsEnumeratedProperty

classCurvesAdjustment ('CrvA')

inherits from: undefined
contains: undefined

Table 9–218: classCurvesAdjustment (6)

Key	Type	Bounds	Options
keyChannel ('Chnl')	typeChannelReference		flagsEnumeratedProperty
keyAuto ('Auto')	typeBoolean		flagsSingleProperty
keyBlackClip ('BlcC')	typeFloat		flagsSingleProperty
keyWhiteClip ('WhtC')	typeFloat		flagsSingleProperty
keyMapping ('Mpng')	typeInteger		flagsListProperty
keyCurve ('Crv ')	classPoint		flagsListProperty

classLevelsAdjustment ('LvlA')

inherits from: none
contains: undefined

Table 9–219: classLevelsAdjustment (7)

Key	Type	Bounds	Options
keyChannel ('Chnl')	typeChannelReference		flagsEnumeratedProperty
keyAuto ('Auto')	typeBoolean		flagsSingleProperty
keyBlackClip ('BlcC')	typeFloat		flagsSingleProperty
keyWhiteClip ('WhtC')	typeFloat		flagsSingleProperty
keyInput ('Inpt')	typeInteger		flagsListProperty
keyGamma ('Gmm ')	typeFloat		flagsSingleProperty
keyOutput ('Otpt')	typeInteger		flagsListProperty

classHueSaturationAdjustment ('HStA')

inherits from: none
contains: undefined

Table 9–220: classHueSaturationAdjustment (4)

Key	Type	Bounds	Options
keyChannel ('Chnl')	typeChannel		flagsEnumeratedProperty
keyHue ('H ')	typeInteger		flagsSingleProperty
keySaturation ('Strt')	typeInteger		flagsSingleProperty
keyLightness ('Lght')	typeInteger		flagsSingleProperty

classHueSaturationAdjustmentV2 ('Hst2')

inherits from: none
contains: undefined

Table 9–221: classHueSaturationAdjustmentV2 (8)

Key	Type	Bounds	Options
keyLocalRange ('LclR')	typeInteger		flagsSingleProperty
keyBeginRamp (BgnR')	typeInteger		flagsSingleProperty
keyBeginSustain ('BgnS')	typeInteger		flagsSingleProperty
keyEndSustain ('EndS')	typeInteger		flagsSingleProperty
keyEndRamp ('EndR')	typeInteger		flagsSingleProperty
keyHue ('H ')	typeInteger		flagsSingleProperty
keySaturation ('Strt')	typeInteger		flagsSingleProperty
keyLightness ('Lght')	typeInteger		flagsSingleProperty

classBlendRange ('Blnd')

inherits from: none
contains: undefined

Table 9–222: classHueSaturationAdjustmentV2 (8)

Key	Type	Bounds	Options
keyChannel ('Chnl')	typeChannelReference		flagsEnumeratedProperty
keySrcBlackMin (SrcB')	typeInteger		flagsSingleProperty
keySrcBlackMax ('Srcl')	typeInteger		flagsSingleProperty
keySrcWhiteMin ('SrcW')	typeInteger		flagsSingleProperty
keySrcWhiteMax ('Srcm')	typeInteger		flagsSingleProperty
keyDestBlackMin ('DstB')	typeInteger		flagsSingleProperty
keyDestBlackMax ('Dstl')	typeInteger		flagsSingleProperty
keyDestWhiteMin ('DstW')	typeInteger		flagsSingleProperty
keyDestWhiteMax ('Dstt')	typeInteger		flagsSingleProperty

classGuide ('Gd ')

inherits from: classElement
contains: undefined

Table 9–223: classGuide (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyPosition ('Pstn')	typeFloat		flagsSingleProperty
keyOrientation ('Ornt')	typeOrientation		flagsEnumeratedProperty

classColorSampler ('ClSm')

inherits from: classElement
contains: undefined

Table 9–224: classColorSampler (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyPosition ('Pstn')	classPoint		flagsSingleProperty

classColorStop ('Clrt')

inherits from: none
contains: undefined

Table 9–225: classColorStop (4)

Key	Type	Bounds	Options
keyLocation ('Lctn')	typeInteger		flagsSingleProperty
keyMidpoint ('Mdpn')	typeInteger		flagsSingleProperty
keyType ('Type')	typeColorStopType		flagsEnumeratedProperty
keyColor ('Clr ')	classColor		flagsSingleProperty

classTransparencyStop ('TrnS')

inherits from: none
contains: undefined

Table 9–226: classTransparencyStop (3)

Key	Type	Bounds	Options
keyLocation ('Lctn')	typeInteger		flagsSingleProperty
keyMidpoint ('Mdpn')	typeInteger		flagsSingleProperty
keyOpacity ('Opct')	unitPercent		flagsSingleProperty

classCalculation ('Clcl')

inherits from: none
contains: undefined

Table 9–227: classCalculation (11)

Key	Type	Bounds	Options
key_Source (keyTo)	typeChannelReference		flagsEnumeratedProperty
keyInvert ('Invr')	typeBoolean		flagsSingleProperty
keyCalculation ('Clcl')	typeCalculation		flagsEnumeratedProperty
keySource2 ('Src2')	typeChannelReference		flagsEnumeratedProperty
keyInvertSource2 ('InvS')	typeBoolean		flagsSingleProperty
keyScale ('Scl ')	typeFloat		flagsSingleProperty
keyOffset ('Ofst')	typeInteger		flagsSingleProperty
keyOpacity ('Opct')	unitPercent		flagsSingleProperty

Table 9–227: classCalculation (11)

Key	Type	Bounds	Options
keyPreserveTransparency ('PrsT')	typeBoolean		flagsSingleProperty
keyUseMask ('UsMs')	typeChannelReference		flagsEnumeratedProperty
keyInvertMask ('InvM')	typeBoolean		flagsSingleProperty

classCustomWhitePoint ('CstW')

inherits from: none
contains: undefined

Table 9–228: classCustomWhitePoint (2)

Key	Type	Bounds	Options
keyX ('X ')	typeFloat		flagsSingleProperty
keyY ('Y ')	typeFloat		flagsSingleProperty

classCustomPhosphors ('CstP')

inherits from: none
contains: undefined

Table 9–229: classCustomPhosphors (6)

Key	Type	Bounds	Options
keyRedX ('RdX ')	typeFloat		flagsSingleProperty
keyRedY ('RdY ')	typeFloat		flagsSingleProperty
keyGreenX ('GrnX')	typeFloat		flagsSingleProperty
keyGreenY ('GrnY')	typeFloat		flagsSingleProperty
keyBlueX ('BlX ')	typeFloat		flagsSingleProperty
keyBlueY ('BlY ')	typeFloat		flagsSingleProperty

classAssumedProfile ('AssP')

inherits from: none
contains: undefined

Table 9–230: classAssumedProfile (2)

Key	Type	Bounds	Options
key_Source ('Srce')	typeAssumeOptions		flagsSingleProperty
keyName ('Nm ')	typeChar		flagsSingleProperty

classXYYColor ('XYYC')

inherits from: none
contains: undefined

Table 9–231: classAssumedProfile (3)

Key	Type	Bounds	Options
keyX ('X ')	typeInteger		flagsSingleProperty
keyY ('Y ')	typeInteger		flagsSingleProperty
keyUpperY ('UppY')	typeInteger		flagsSingleProperty

classPoint16 ('Pnt1')

inherits from: none
contains: undefined

Table 9–232: classPoint16 (2)

Key	Type	Bounds	Options
keyHorizontal ('Hrzn')	typeInteger		flagsSingleProperty
keyVertical ('Vrtc')	typeInteger		flagsSingleProperty

classRGBSetup ('RGBt')

inherits from: none
contains: undefined

Table 9–233: classRGBSetup (6)

Key	Type	Bounds	Options
key_Source ('Srce')	typeRGBSetupSource		flagsEnumeratedProperty
keyName ('Nm ')	typeStringFSS		flagsSingleProperty
keyGamma ('Gmm ')	typeFloat		flagsSingleProperty
keyWhitePoint ('WhtP')	typeKelvinCustomWhite-Point		flagsSingleProperty
keyPhosphors ('Phsp')	typePhosphorsCustom-Phosphors		flagsSingleProperty
keyCompensation ('Cmpn')	typeBoolean		flagsEnumeratedProperty

classCMYKSetup ('CMYS')

inherits from: none
contains: undefined

Table 9–234: classCMYKSetup (17)

Key	Type	Bounds	Options
keyEngine ('Engn')	typeCMYKSetupEngine		flagsEnumeratedProperty
keyUsing ('Usng')	typeFSS		flagsSingleProperty
keyUseICCProfile ('UsIC')	typeBoolean		flagsEnumeratedProperty
keyInkColors ('Clrs')	typeChar		flagsSingleProperty

Table 9–234: classCMYKSetup (17)

Key	Type	Bounds	Options
keyColorsList ('ClrL')	classXYYColor		flagsListProperty
keyDotGain ('DtGn')	unitPercent		flagsSingleProperty
keyDotGainCurves ('DtGC')	typeRawData		flagsSingleProperty
keyGCR ('GCR ')	typeBoolean		flagsEnumeratedProperty
keyBlackLimit ('BlcL')	unitPercent		flagsSingleProperty
keyTotalLimit ('TtlL')	unitPercent		flagsSingleProperty
keyUCA ('UC ')	unitPercent		flagsSingleProperty
keyBlackGenerationCurve ('BlcG')	classPoint16		flagsListProperty
keyBlackGeneration ('Blcn')	typeBlackGeneration		flagsSingleProperty
keyICCEngine ('ICCE')	typeChar		flagsSingleProperty
keyICCSetupName ('ICCt')	typeChar		flagsSingleProperty
keyIntent ('Inte')	typeIntent		flagsEnumeratedProperty
keyMapBlack ('MpBl')	typeBoolean		flagsEnumeratedProperty

classGraySetup ('GrSt')

inherits from: none
contains: undefined

Table 9–235: classGraySetup (1)

Key	Type	Bounds	Options
keyGrayBehavior ('GrBh')	typeGrayBehavior		flagsEnumeratedProperty

classProfileSetup ('PrfS')

inherits from: none
contains: undefined

Table 9–236: classProfileSetup (10)

Key	Type	Bounds	Options
keyEmbedRGB ('EmbR')	typeBoolean		flagsEnumeratedProperty
keyEmbedCMYK ('EmbC')	typeBoolean		flagsEnumeratedProperty
keyEmbedGray ('EmbG')	typeBoolean		flagsEnumeratedProperty
keyEmbedLab ('EmbL')	typeBoolean		flagsEnumeratedProperty
keyMismatchRGB ('MsmR')	typeProfileMismatch		flagsEnumeratedProperty
keyMismatchCMYK ('MsmC')	typeProfileMismatch		flagsEnumeratedProperty
keyMismatchGray ('MsmG')	typeProfileMismatch		flagsEnumeratedProperty
keyAssumedRGB ('AssR')	classAssumedProfile		flagsSingleProperty
keyAssumedCMYK ('AssC')	classAssumedProfile		flagsSingleProperty
keyAssumedGray ('AssG')	classAssumedProfile		flagsSingleProperty

classGradientTool ('GrTl')

inherits from: classTool
contains: undefined

Table 9–237: classGradientTool (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classTool		flagsSingleProperty

classHistoryState ('HstS')

inherits from: classElement
contains: undefined

Table 9–238: classHistoryState (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty
keyName ('Nm ')	typeText		flagsSingleProperty
keyUsing ('Usng')	typeHistoryStateSource		flagsEnumeratedProperty

classSnapshot ('SnpS')

inherits from: classHistoryState
contains: undefined

Table 9–239: classSnapshot (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classHistoryState		flagsSingleProperty

classFormat ('Fmt ')

inherits from: none
contains: undefined

Table 9–240: classFormat (1)

Key	Type	Bounds	Options
keyFormat ('Fmt ')	typeClassFormat		flagsEnumeratedProperty

classImport ('Impr')

inherits from: none
contains: undefined

Table 9–241: classImport (1)

Key	Type	Bounds	Options
keyImport ('Impr')	typeClassImport		flagsEnumeratedProperty

classExport ('Expr')

inherits from: none
contains: undefined

Table 9–242: classExport (1)

Key	Type	Bounds	Options
keyExport ('Expr')	typeClassExport		flagsEnumeratedProperty

classMenuItem ('Mn ')

inherits from: classElement
contains: undefined

Table 9–243: classMenuItem (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classElement		flagsSingleProperty

Formats

classJPEGFormat ('JPEG')

inherits from: classFormat
contains: undefined

Table 9–244: classJPEGFormat (5)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat		flagsSingleProperty
keyQuality ('Qlty ')	typeInteger		flagsSingleProperty
keyOptimized ('Optm')	typeBoolean		flagsEnumeratedProperty
keyScans ('Scns')	typeInteger		flagsSingleProperty
keySavePaths ('SvPt')	typeBoolean		flagsSingleProperty

classPhotoshopEPSFormat ('PhtE')

inherits from: classFormat
contains: undefined

Table 9–245: classPhotoshopEPSFormat (11)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat		flagsSingleProperty
keyPreview ('Prvw ')	typeEPSPreview		flagsEnumeratedProperty
keyDepth ('Dpth')	typeDepth		flagsEnumeratedProperty
keyEncoding ('Encd')	typeEncoding		flagsEnumeratedProperty
keyQuality ('Qlty')	typeQuality		flagsEnumeratedProperty
keyClippingPathEPS ('ClpP')	typeChar		flagsSingleProperty
keyFlatness ('Fltn ')	typeFloat		flagsSingleProperty
keyHalftoneScreen ('HlfS')	typeBoolean		flagsSingleProperty
keyTransferFunction ('TrnF')	typeBoolean		flagsSingleProperty
keyColorManagement ('ClMg')	typeBoolean		flagsSingleProperty
keyTransparentWhites ('TrnW')	typeBoolean		reserved, singleItem, notEnumerated, readWrite, reserved, reserved, reserved, reserved, reserved, reserved, reserved, reserved, noApostrophe, notFeminine, notMasculine, singular

classPICTFileFormat ('PICF')

inherits from: classFormat

contains: undefined

Table 9–246: classPICTFileFormat (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('Fmt ')		flagsSingleProperty
keyResolution ('Rslt')	typeDeepDepth ('DpDp')		flagsEnumeratedProperty
keyJPEGQuality ('JPEQ')	typeQuality ('Qlty')		flagsEnumeratedProperty

classRawFormat ('Rw ')

inherits from: classFormat

contains: undefined

Table 9–247: classRawFormat (12)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('Fmt ')		flagsSingleProperty
keyWidth ('Wdth')	typeInteger		flagsSingleProperty
keyHeight ('Hght')	typeInteger		flagsSingleProperty
keyChannels ('Chns')	typeInteger		flagsSingleProperty
keyDepth ('Dpth')	typeInteger		flagsSingleProperty
keyRetainHeader ('RtnH')	typeBoolean		flagsSingleProperty
keyFileType ('FlTy')	typeText ('typeChar')		flagsSingleProperty
keyFileCreator ('FlCr')	typeText ('typeChar')		flagsSingleProperty
keyHeader ('Hdr ')	typeInteger		flagsSingleProperty
keyOriginalHeader ('OrgH')	typeBoolean		flagsSingleProperty
keyChannelsInterleaved ('ChnI')	typeBoolean		flagsSingleProperty
keyByteOrder ('ChnI')	typePlatform ('Pltf')		flagsEnumeratedProperty

classTIFFFormat ('TIFF')

inherits from: classFormat

contains: unknown

Table 9–248: classTIFFFormat (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('Fmt ')		flagsSingleProperty
keyByteOrder ('BytO')	typePlatform ('Pltf')		flagsEnumeratedProperty
keyLZWCompression ('LZWC')	typeBoolean		flagsSingleProperty

classPICTResourceFormat ('PICR')

inherits from: classPICTFileFormat

contains: undefined

Table 9–249: classPICTResourceFormat (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classPICTFileFormat ('PICF')		flagsSingleProperty
keyResourceID ('Rsrl')	typeInteger		flagsSingleProperty
keyName ('Nm ')	typeText ('typeChar')		flagsSingleProperty

classGIFFormat ('GFFr')

inherits from: classFormat
contains: undefined

Table 9–250: classGIFFormat (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat		flagsSingleProperty
keyInterlace ('Intr')	typeBoolean		flagsSingleProperty

classGIF89aExport ('GF89')

inherits from: classImport
contains: undefined

Table 9–251: classGIF89aExport (16)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classImport)		flagsSingleProperty
keyGIFRequiredColor-SpaceType ('GFCS')	typeGIFRequiredColor-SpaceType		flagsEnumeratedProperty
keyGIFRowOrderType ('GFIT')	typeGIFRowOrderType		flagsEnumeratedProperty
keyGIFPaletteType (GFPL')	typeGIFPaletteType		flagsEnumeratedProperty
keyGIFPaletteFile ('GFPF')	typeAlias		flagsSingleProperty
keyGIFColorFileType ('GFPT')	typeGIFColorFileType		flagsEnumeratedProperty
keyGIFColorLimit ('GFCL')	typeInteger		flagsSingleProperty
keyGIFUseBestMatch ('GFBM')	typeBoolean		flagsSingleProperty
keyGIFExportCaption ('GFEC')	typeBoolean		flagsSingleProperty
keyGIFTransparentIn-dexRed ('GFTR')	typeInteger		flagsSingleProperty
keyGIFTransparentIn-dexGreen ('GFTG')	typeInteger		flagsSingleProperty
keyGIFTransparentIndex-Blue ('GFTB')	typeInteger		flagsSingleProperty

Table 9–251: classGIF89aExport (16)

Key	Type	Bounds	Options
keyGIFMaskChannelIndex ('GFMI')	typeInteger		flagsSingleProperty
keyGIFMaskChannelInverted ('GFMV')	typeBoolean		flagsSingleProperty
keyGIFTransparentColor ('GFTC')	classRGBColor		flagsSingleProperty
keyIn ('In ')	typePlatformFilePath		flagsSingleProperty

classTargaFormat ('TrgF')

inherits from: classFormat
contains: undefined

Table 9–252: classTargaFormat (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat		flagsSingleProperty
keyBitDepth ('BtDp')	typeInteger		flagsSingleProperty

classBMPFormat ('BMPF')

inherits from: classFormat
contains: undefined

Table 9–253: classBMPFormat (4)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat		flagsSingleProperty
keyPlatform ('BtDp')	typePlatform		flagsEnumeratedProperty
keyBitDepth ('BtDp')	typeBitDepth		flagsEnumeratedProperty
keyCompression ('BtDp')	typeBoolean		flagsSingleProperty

classPhotoshop20Format ('Pht2')

inherits from: classFormat

contains: undefined

Table 9–254: classPhotoshop20Format (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('#CIF')		flagsSingleProperty

classPhotoshop35Format ('Pht3')

inherits from: classFormat

contains: undefined

Table 9–255: classPhotoshop35Format (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('#ClF')		flagsSingleProperty

classPhotoshopDCS2Format ('PhD2')

inherits from: classPhotoshopEPSFormat

contains: undefined

Table 9–256: classPhotoshopDCS2Format (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classPhotoshopEPSFormat ('PhtE')		flagsSingleProperty
keyDCS ('DCS ')	typeDCS ('DCS ')		flagsEnumeratedProperty

classPhotoshopDCSFormat ('PhD1')

inherits from: classPhotoshopEPSFormat

contains: undefined

Table 9–257: classPhotoshopDCSFormat (2)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classPhotoshopEPSFormat ('PhtE')		flagsSingleProperty
keyDCS ('DCS ')	typeDCS ('DCS ')		flagsEnumeratedProperty

classPhotoshopPDFFormat ('PhtP')

inherits from: classFormat

contains: undefined

Table 9–258: classPhotoshopPDFFormat (3)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('#ClF')		flagsSingleProperty
keyQuality ('Qlty')	typeInteger		flagsSingleProperty
keyEncoding ('Encd')	typeEncoding ('Encd')		flagsEnumeratedProperty

classScitexCTFormat ('Sctx')

inherits from: format

contains: undefined

Table 9–259: classScitexCTFormat (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('#ClF')		flagsSingleProperty

classEPSPICTPreview ('EPSC')

inherits from: format

contains: undefined

Table 9–260: classEPSPICTPreview (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('#ClF')		flagsSingleProperty

classEPSTIFFPreview ('EPST')

inherits from: format

contains: undefined

Table 9–261: classEPSTIFFPreview (1)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('#ClF')		flagsSingleProperty

classEPSGenericFormat ('EPSG')

inherits from: classFormat

contains: undefined

Table 9–262: classEPSGenericFormat (7)

Key	Type	Bounds	Options
keyInherits ('c@#^')	classFormat ('#ClF')		flagsSingleProperty
keyWidth ('Wdth')	unitDistance('#Rlt'')		flagsEnumeratedProperty
keyHeight ('Hght')	unitDistance ('#Rlt'')		flagsEnumeratedProperty
keyResolution ('Rslt')	unitDensity ('#Rsl')		flagsEnumeratedProperty
keyMode ('Md ')	typeColorSpace ('ClrS')		flagsEnumeratedProperty
keyAntiAlias ('AntA')	typeBoolean		flagsSingleProperty
keyConstrainProportions ('CnsP')	typeBoolean		flagsSingleProperty

12. Types and Enumerations

This chapter describes of all the types and enumerations used in scripting Photoshop.

Types and Enumerations

Type Name	Enumerations
typeActionReference (' #Act ')	
typeAlignDistributeSelector ('ADSt')	enumADSTops ('AdTp') enumADSCentersV ('AdCV') enumADSBottoms ('AdBt') enumADSVertical ('AdVr') enumADSLefts ('AdLf') enumADSCentersH ('AdCH') enumADSRights ('AdRg') enumADSHorizontal ('AdHr')
typeAlignment ('Alg ')	enumLeft ('Left') enumCenter ('Cntr') enumRight ('Rght') enumJustifyFull ('JstF') enumJustifyAll ('JstA')
typeAreaSelector (' ArSl ')	enumSelection ('Slct') enumImage ('Img ')
typeAssumeOptions (' AssO ')	enumNone ('None') enumAskWhenOpening ('AskW') enumMonitor ('Moni') enumBuiltin ('BItn') enumICC ('ICC ')
typeBevelEmbossStampStyle (' BESs ')	enumStampIn ('In ') enumStampOut ('Out ')
typeBevelEmbossStyle (' BESl ')	enumOuterBevel ('OtrB') enumInnerBevel ('InrB') enumEmboss ('Embs') enumPillowEmboss ('PlEb')
typeBitDepth (' BtDp ')	enumBitDepth1 ('BD1 ') enumBitDepth4 ('BD4 ') enumBitDepth8 ('BD8 ') enumBitDepth24 ('BD24')
typeBlackGeneration (' BlcG ')	enumNone ('None') enumLight ('Lgt ') enumMedium ('Mdim') enumHeavy ('Hvy ') enumMaximum ('Mxmm')

Type Name	Enumerations
typeBlendMode('BlnM')	enumNormal ('Nrml') enumDissolve ('Dslv') enumBehind ('Bhnd') enumClear ('Clar') enumMultiply ('Mltp') enumScreen ('Scrn') enumOverlay ('Ovrl') enumSoftLight ('SftL') enumHardLight ('HrdL') enumDarken ('Drkn') enumLighten ('Lghn') enumDifference ('Dfrn') enumHue ('H ') enumSaturation ('Strt') enumColor ('Clr ') enumLuminosity ('Lmns') enumExclusion ('Xclu') enumColorDodge ('CDdg') enumColorBurn ('CBrn')
typeBlurMethod ('BlrM')	enumSpin ('Spn ')
typeBlurQuality ('BlrQ')	enumDraft ('Drft') enumGood ('Gd ') enumBest ('Bst ')
typeBrushType ('BrsT')	enumBrushSimple ('BrSm') enumBrushLightRough ('BrsL') enumBrushDarkRough ('BrDR') enumBrushWideSharp ('BrsW') enumBrushWideBlurry ('BrbW')
typeBuiltinProfile ('BltP')	enumAppleRGB ('AppR') enumSRGB ('SRGB') enumCIERGB ('CRGB') enumNTSC ('NTSC') enumPalSecam ('PlSc') enumGray18 ('Gr18') enumGray22 ('Gr22')
typeCMYKSetupEngine ('CMYE')	enumBuiltin ('Bltn') enumICC ('ICC ' enumTables ('Tbl ')
typeCalculation ('Clcn')	enumNormal ('Nrml') enumMultiply ('Mltp') enumScreen ('Scrn') enumOverlay ('Ovrl') enumSoftLight ('SftL') enumHardLight ('HrdL') enumDarken ('Drkn') enumLighten ('Lghn') enumDifference ('Dfrn') enumExclusion ('Xclu') enumColorDodge ('CDdg') enumColorBurn ('CBrn') enumAdd ('Add ') enumSubtract ('Sbtr')

Type Name	Enumerations
typeChannel ('Chnl')	enumBlack ('Blck') enumCMYK ('CMYK') enumRGB ('RGB ') enumLab ('Lab ') enumRed ('Rd ') enumGreen ('Grn ') enumCyan ('Cyn ') enumLightness ('Lght') enumBlue ('Bl ') enumMagenta ('Mgnt') enumYellow ('Yllw') enumA ('A ') enumB ('B ') enumComposite ('Cmps') enumMask ('Msk ') enumMonotone ('Mntn') enumDuotone ('Dtn ') enumTritone ('Trtn') enumQuadtone ('Qdtn') enumTransparency ('Trsp')
typeChannelReference ('#ChR')	
typeClass (typeType)	
typeClassColor ('#Clr')	
typeClassElement ('#ClE')	
typeClassExport ('#Cle')	
typeClassFormat ('#ClF')	
typeClassHueSatHueSatV2 ('#HsV')	
typeClassImport ('#Cll')	
typeClassMode ('#ClM')	
typeClassStringFormat ('#ClS')	
typeClassTextExport ('#CTE')	
typeClassTextImport ('#ClT')	
typeColor('Clr ')	enumRed ('Rd ') enumOrange ('Orng') enumYellowColor ('Ylw ') enumGreen ('Grn ') enumBlue ('Bl ') enumViolet ('Vlt ') enumGray ('Gry ')
typeColorPalette ('ClrP')	enumExact ('Exct') enumWeb ('Web ') enumUniform ('Unfm') enumAdaptive ('Adpt') enumPrevious ('Prvs') enumSpectrum ('Spct') enumGrayscale ('Grys') enumBlackBody ('BlcB') enumMacintoshSystem ('McnS') enumWindowsSystem ('WndS')

Type Name	Enumerations
typeColorSpace ('ClrS')	enumGrayscale ('Grys') enumRGBColor ('RGBC') enumCMYKColor ('ECMY') enumLabColor ('LbCl') enumBitmap ('Btmp') enumGrayScale ('Gryc') enumGray16 ('GryX') enumIndexedColor ('Indl') enumRGB48 ('RGBF') enumCMYK64 ('CMSF') enumHSLColor ('HSLC') enumHSBColor ('HSBl') enumMultichannel ('Mlth') enumLab48 ('LbCF')
typeColorStopType ('Clry')	enumForegroundColor ('FrgC') enumBackgroundColor ('BckC') enumUserStop ('UsrS')
typeColors ('Clrs')	enumReds ('Rds ') enumYellows ('Ylws') enumGreens ('Grns') enumCyans ('Cyns') enumBlues ('Bls ') enumMagentas ('Mgnt') enumWhites ('Whts') enumNeutrals ('Ntrl') enumBlacks ('Blks') enumHighlights ('Hghl') enumMidtones ('Mdtn') enumShadows ('Shdw') enumOutOfGamut ('OtOf')
typeCompensation ('Cmpn')	enumNone ('None') enumBuiltin ('Bltn')
typeContourEdge ('CntE')	enumUpper ('Upr ') enumLower ('Lwr ')
typeConvert ('Cnvr')	enumRectToPolar ('RctP') enumPolarToRect('PlrR')
typeCorrectionMethod ('CrcM')	enumRelative ('Rltv') enumAbsolute ('Absl')
typeDCS ('DCS ')	enumSingleNoCompositePS ('NCmS') enumSingle72Gray ('72GS') enumSingle72Color ('72CS') enumMultiNoCompositePS ('NCmM') enumMulti72Gray ('72GM') enumMulti72Color ('72CM') enumNoCompositePS ('NCmp') enum72Gray ('72Gr') enum72Color ('72Cl')
typeDeepDepth ('DpDp')	enum2BitsPerPixel ('2Bts') enum4BitsPerPixel ('4Bts') enum8BitsPerPixel ('EghB') enum16BitsPerPixel ('16Bt') enum32BitsPerPixel ('32Bt')
typeDepth ('Dpth')	enum1BitPerPixel ('OnBt') enum8BitsPerPixel ('EghB')
typeDiffuseMode ('DfsM')	enumNormal ('Nrml') enumLightenOnly ('LghO') enumDarkenOnly ('DrkO')
typeDirection ('Drct')	enumLeft ('Left') enumRight ('Rght')
typeDisplacementMap ('DspM')	enumStretchToFit ('StrF') enumTile ('Tile')

Type Name	Enumerations
typeDistribution ('Dstr')	enumUniformDistribution ('Unfr') enumGaussianDistribution ('Gsn ')
typeDither ('Dthr')	enumPattern ('Ptrn') enumDiffusion ('Dfsn')
typeDitherQuality ('Dthq')	enumBetter ('Dthb') enumFaster ('Dthf')
typeDocumentReference ('#DcR')	
typeEPSPreview ('EPSP')	enumTIFF ('TIFF') enumMacintosh ('Mcnt')
typeElementReference ('#ElR')	
typeEncoding ('Encd')	enumASCII ('ASCI') enumBinary ('Bnry') enumJPEG ('JPEG') enumZip ('ZpEn')
typeExtrudeRandom ('ExtR')	enumRandom ('Rndm') enumLevelBased ('LvlB')
typeExtrudeType ('ExtT')	enumBlocks ('Blks') enumPyramids ('Pyrm')
typeEyeDropperSample ('EyDp')	enumSamplePoint (SmpP) enumSample3x3 (Smp3) enumSample5x5 (Smp5)
typeFPXCompress ('FxCm')	enumFPXCompressNone ('FxNo') enumFPXCompressLossyJPEG ('FxJP')
typeFill ('Fl ')	enumWhite ('Wht ') enumBackgroundColor ('BckC') enumTransparent ('Trns')
typeFillColor ('FlCl')	enumFillBack ('FlBc') enumFillFore ('FlFr') enumFillInverse ('Flln') enumFillSame ('FlSm')
typeFillContents ('FlCn')	enumForegroundColor ('FrgC') enumBackgroundColor ('BckC') enumPattern ('Ptrn') enumSaved ('Sved') enumSnapshot ('Snps') enumBlack ('Blck') enumWhite ('Wht ') enumGray ('Gry ')
typeFillMode ('FlMd')	enumBackground ('Bckg') enumRepeat ('Rpt ') enumWrap ('Wrp ')
typeGIFColorFileType ('GFPT')	enumGIFColorFileColors ('GFCF') enumGIFColorFileColorTable ('GFCT') enumGIFColorFileMicrosoftPalette ('GFMS')
typeGIFPaletteType ('GFPL')	enumGIFPaletteExact ('GFPE') enumGIFPaletteAdaptive ('GFPA') enumGIFPaletteSystem ('GFPS') enumGIFPaletteOther ('GFPO')
typeGIFRequiredColorSpaceType ('GFCS')	enumGIFRequiredColorSpaceRGB ('GFRG') enumGIFRequiredColorSpaceIndexed ('GFCI')
typeGIFRowOrderType ('GFIT')	
typeGlobalClass ('GlbC')	enumGIFRowOrderNormal ('GFNI') enumGIFRowOrderInterlaced ('GFIN')
typeGlobalObject ('GlbO')	

Type Name	Enumerations
typeGradientType ('GrdT')	enumLinear ('Lnr ') enumRadial ('Rdl ') enumAngle ('Angl') enumReflected ('Rflc') enumDiamond ('Dmnd')
typeGrainType ('Grnt')	enumGrainRegular ('GrnR') enumGrainSoft ('GrSf') enumGrainSprinkles ('GrSr') enumGrainClumped ('GrnC') enumGrainContrasty ('GrCn') enumGrainEnlarged ('GrnE') enumGrainStippled ('GrSt') enumGrainHorizontal ('GrnH') enumGrainVertical ('GrnV') enumGrainSpeckle ('GrSp')
typeGrayBehavior ('GrBh')	enumRGB ('RGB ') enumBlack ('Blck')
typeHistoryStateSource ('HstS')	enumFullDocument ('FlID') enumMergedLayers ('MrgL') enumCurrentLayer ('CrrL')
typeHorizontalLocation ('HrzL')	enumLeft ('Left') enumRight ('Rght')
typeImageReference ('#ImR')	
typeInnerGlowSource ('IGSr')	enumCenterGlow ('SrcC') enumEdgeGlow ('SrcE')
typeIntent ('Inte')	enumImage ('Img ') enumGraphics ('Grp ') enumColorimetric ('Clrm')
typeInterlaceCreateType ('IntC')	enumCreateDuplicate ('CrtD') enumCreateInterpolation ('CrtI')
typeInterlaceEliminateType ('IntE')	enumEliminateOddFields ('ElmO') enumEliminateEvenFields ('ElmE')
typeInterpolation ('Intp')	enumNearestNeighbor ('Nrst') enumBilinear ('Blnr') enumBicubic ('Bcbc')
typeKelvin ('Klvn')	enum5000 ('5000') enum5500 ('5500') enum6500 ('6500') enum7500 ('7500') enum9300 ('9300') enumStdA ('StdA') enumStdB ('StdB') enumStdC ('StdC') enumStdE ('StdE')
typeKelvinCustomWhitePoint ('#Klv')	
typeLens ('Lns ')	enumZoom ('Zm ') enumPanaVision ('PnVs') enumNikon ('Nkn ') enumNikon105 ('Nkn1')
typeLightDirection ('LghD')	enumLightDirBottom ('LDBt') enumLightDirBottomLeft ('LDBL') enumLightDirLeft ('LDLf') enumLightDirTopLeft ('LDTL') enumLightDirTop ('LDTp') enumLightDirTopRight ('LDTR') enumLightDirRight ('LDRg') enumLightDirBottomRight ('LDBR')

Type Name	Enumerations
typeLightPosition ('LghP')	enumLightPosBottom ('LPBt') enumLightPosBottomLeft ('LPBL') enumLightPosBottomRight ('LPBr') enumLightPosLeft ('LPLf') enumLightPosTopLeft ('LPTL') enumLightPosTop ('LPTp') enumLightPosTopRight ('LPTR') enumLightPosRight ('LPRg')
typeLightType ('LghT')	enumLightDirectional ('LghD') enumLightOmni ('LghO') enumLightSpot ('LghS')
typeLocationReference ('#Lct')	
typeMaskIndicator ('Mskl')	enumMaskedAreas ('MskA') enumSelectedAreas ('SlcA') enumSpotColor ('Spot')
typeMenuItem ('MnIt')	enumAboutAp ('AbAp') enumPlace ('Plce') enumOpenAs ('OpAs') enumFileInfo ('FlIn') enumPageSetup ('PgSt')
typeMethod ('Mthd')	enumThreshold ('Thrh') enumPatternDither ('PtnD') enumDiffusionDither ('DfnD') enumHalftoneScreen ('HlfS') enumHalftoneFile ('HlfF') enumCustomPattern ('Cstm')
typeMezzotintType ('MztT')	enumFineDots ('FnDt') enumMediumDots ('MdmD') enumGrainyDots ('GrnD') enumCoarseDots ('CrsD') enumShortLines ('ShrL') enumMediumLines ('MdmL') enumLongLines (LngL) enumShortStrokes ('ShSt') enumMediumStrokes ('MdmS') enumLongStrokes ('LngS')
typeMode ('Md ')	enumModeGray ('MdGr') enumModeRGB ('MdRG')
typeObject ('Objc')	
typeObjectReference (typeObject Specifier)	
typeOnOff ('OnOf')	enumOn ('On ') enumOff ('Off ')
typeOrdinal ('Ordn')	enumAll ('Al ') enumFirst ('Frst') enumLast ('Lst ') enumNext ('Nxt ') enumPrevious ('Prvs') enumMiddle ('Mddl') enumAny ('Any ') enumNone ('None') enumTarget ('Trgt') enumForward ('Frwr') enumBackward ('Bckw') enumFront ('Frnt') enumBack ('Back') enumMerged ('Mrgd') enumLinked ('Lnkd')
typeOrientation ('Ornt')	enumHorizontal ('Hrzn') enumVertical ('Vrtc')

Type Name	Enumerations
typePNGFilter ('PNGf')	enumPNGFilterNone ('PGNo') enumPNGFilterSub ('PGSb') enumPNGFilterUp ('PGUp') enumPNGFilterAverage ('PGAv') enumPNGFilterPaeth ('PGPt') enumPNGFilterAdaptive ('PGAd')
typePNGInterlaceType ('PGIT')	enumPNGInterlaceNone ('PGIN') enumPNGInterlaceAdam7 ('PGIA')
typePagePosition ('PgPs')	enumPagePosTopLeft ('PgTL') enumPagePosCentered ('PgPC')
typePathReference ('#PtR')	
typePhosphors ('Phsp')	enumCIERGB ('CRGB') enumEBUITU ('EBT ') enumHDTV ('HDTV') enumNTSC ('NTSC') enumP22EBU ('P22B') enumSMPTE240M ('SMPT') enumSMPTEC ('SMPC') enumTrinitron ('Trnt')
typePhosphorsCustomPhosphors ('#Phs')	
typePickerKind ('PckK')	enumSystemPicker ('SysP') enumPhotoshopPicker ('Phtk') enumPluginPicker ('PlgP')
typePixelPaintSize ('PPSz')	enumPixelPaintSize1 ('PxS1') enumPixelPaintSize2 ('PxS2') enumPixelPaintSize3 ('PxS3') enumPixelPaintSize4 ('PxS4')
typePlatform ('Pltf')	enumOS2 ('OS2 ') enumWindows ('Win ') enumMacintosh ('Mcnt') enumIBMPC ('IBMP')
typePreview ('Prvw')	enumNone ('None') enumIcon ('Icn ') enumThumbnail ('Thmb') enumMacThumbnail ('McTh') enumWinThumbnail ('WnTh') enumFullSize ('FlSz')
typePreviewCMYK ('Prvt')	enumPreviewOff ('PrvO') enumPreviewCMYK ('PrvC') enumPreviewCyan ('Prvy') enumPreviewMagenta ('PrvM') enumPreviewYellow ('PrvY') enumPreviewBlack ('PrvB') enumPreviewCMY('PrvN')
typeProfileMismatch ('PrfM')	enumIgnore ('Ignr') enumAskWhenOpening ('AskW') enumConvertToCMYK ('CnvC') enumConvertToRGB ('CnvR') enumConvertToLab ('CnvL') enumConvertToGray ('CnvG')
typePurgeItem ('Prgl')	enumClipboard ('Clpb') enumSnapshot ('Snps') enumUndo ('Und ') enumPattern ('Ptrn') enumHistory ('Hsty') enumAll ('Al ')

Type Name	Enumerations
typeQuadCenterState ('QCSt')	enumQCSAverage ('Qcsa') enumQCSCorner0 ('Qcs0') enumQCSCorner1 ('Qcs1') enumQCSCorner2 ('Qcs2') enumQCSCorner3 ('Qcs3') enumQCSSide0 ('Qcs4') enumQCSSide1 ('Qcs5') enumQCSSide2 ('Qcs6') enumQCSSide3 ('Qcs7') enumQCSIndependent ('Qcsi')
typeQuality ('Qlty')	enumLowQuality ('Lw ') enumMediumQuality ('Mdm ') enumHighQuality ('Hgh ') enumMaximumQuality ('Mxm ')
typeRGBSetupSource ('RGBS')	enumCustom ('Cst ') enumBuiltin ('Bltn') enumMonitor ('Moni') enumFile ('Fle ')
typeRawData ('tdta')	
typeRippleSize ('RplS')	enumSmall ('Sml ') enumMedium ('Mdim') enumLarge ('Lrg ')
typeRulerUnits ('RlrU')	enumRulerPixels ('RrPx') enumRulerInches ('RrIn') enumRulerCm ('RrCm') enumRulerPoints ('RrPt') enumRulerPicas ('RrPi') enumRulerPercent ('RrPr')
typeScreenType ('ScrT')	enumScreenCircle ('ScrC') enumScreenDot ('ScrD') enumScreenLine ('ScrL')
typeShape ('Shp ')	enumRound ('Rnd ') enumDiamond ('Dmnd') enumEllipse ('Elps') enumLine ('Ln ') enumSquare ('Sqr ') enumCross ('Crs ')
typeSmartBlurMode ('SmBM')	enumSmartBlurModeNormal ('SBMN') enumSmartBlurModeEdgeOnly ('SBME') enumSmartBlurModeOverlayEdge ('SBMO')
typeSmartBlurQuality ('SmBQ')	enumSmartBlurQualityLow ('SBQL') enumSmartBlurQualityMedium ('SBQM') enumSmartBlurQualityHigh ('SBQH')
typeSpherizeMode ('SphM')	enumNormal ('Nrml') enumHorizontalOnly ('HrzO') enumVerticalOnly ('VrtO')
typeState ('Stte')	enumRedrawComplete ('RdCm')
typeStringClassFormat ('#StC')	
typeStringCompensation ('#Stm')	
typeStringFSS ('#Stf')	
typeStringInteger ('#Stl ')	
typeStrokeDirection ('StrD')	enumStrokeDirRightDiag ('SDRD') enumStrokeDirHorizontal ('SDHz') enumStrokeDirLeftDiag ('SDLD') enumStrokeDirVertical ('SDVt')
typeStrokeLocation ('StrL')	enumInside ('Insd') enumOutside ('Otsd') enumCenter ('Cntr')

Type Name	Enumerations
typeText (typeChar)	
typeTextureType ('TxtT')	enumTexTypeBrick ('TxBr') enumTextTypeBurlap ('TxBu') enumTextTypeCanvas ('TxCa') enumTexTypeSandstone ('TxSt') enumTexTypeBlocks ('TxBl') enumTexTypeFrosted ('TxFr') enumTexTypeTinyLens ('TxTL')
typeTypeClassModeOrClassMode ('#TyM')	
typeUndefinedArea ('UndA')	enumWrapAround ('WrpA') enumRepeatEdgePixels ('RptE')
typeUnitFloat ('UntF')	
typeUrgency ('Urgn')	enumLow ('Low ') enumHigh ('High')
typeUserMaskOptions ('UsrM')	enumHideAll ('HdAl') enumRevealAll ('RvlA') enumHideSelection ('HdSl') enumRevealSelection ('RvlS')
typeValueList ('VlLs')	
typeVerticalLocation ('VrtL')	enumTop ('Top ') enumBottom ('Bttm')
typeWaveType ('Wvtp')	enumWaveSine ('WvSn') enumWaveTriangle ('WvTr') enumWaveSquare ('WvSq')
typeWindMethod ('WndM')	enumWind ('Wnd ') enumBlast ('Blst') enumStagger ('Stgr')
typeYesNo ('YsN ')	enumYes ('Ys ') enumNo ('N ') enumAsk ('Ask ')
typeZigZagType ('ZZTy')	

13. Automation Plug-ins

This chapter lists parameters for automation plug-ins.

eventContactSheet ("63676b34-cb65-11d1-bc43-0060b0a13dc4")

Table 11–1: eventContactSheet Parameters(8)

Key	Type	Bounds	Options
keyInputDirectory ('InpD')	typeAlias		flagsSingleParameter
keySheetWidth ('Wdth')	typeUnitFloat ('UntF')		flagsSingleParameter
keySheetHeight ('Hght')	typeUnitFloat ('UntF')		flagsSingleParameter
keyResolution ('RsIt')	typeUnitFloat ('UntF')		flagsSingleParameter
keyMode ('Md ')	classType		flagsSingleParameter
keyRowOrder ('RowO')	typeBoolean		flagsSingleParameter
keyNumberofCol-umns ('Cols')	typeInteger		flagsSingleParameter
keyNumberofRows ('Rows')	typeInteger		flagsSingleParameter

eventExportTransparentImage ("02879e00-cb66-11d1-bc43-0060b0a13dc4")

Table 11–2: eventExportTransparentImage Parameters(2)

Key	Type	Bounds	Options
keyWidth('Wdth')	typeUnitFloat		flagsSingleParameter
keyHeight ('Hght')	typeUnitFloat		flagsSingleParameter

eventFitImage

("3caa3434-cb67-11d1-bc43-0060b0a13dc4")

Table 11–3: eventFitImage Parameters(4)

Key	Type	Bounds	Options
keyWidth('Wdth')	typeUnitFloat		flagsSingleParameter
keyHeight ('Hght')	typeUnitFloat		flagsSingleParameter

eventModeChange ("8cba8cd6-cb66-11d1-bc43-0060b0a13dc4")

Table 11–4: eventModeChange Parameters(3)

Key	Type	Bounds	Options
keySourceMode ('SrcM')	TypeSource-Mode('CnDn'	enumUIBitmap, enumUIGrayscale enumUIDuotone enumUIIndexed enumUIRGB enumUICMYK enumUILab enumUIMultichannel	flagsListParameter
keyDestination-Mode('DstM')	classType		flagsSingleParameter
keyFlatten('Fltt')	typeBoolean		flagsSingleParameter

eventMultiPagePDFtoPSD ("ec8d7010-cb66-11d1-bc43-0060b0a13dc4")

Table 11–5: eventMultiPagePDFtoPSD Parameters(9)

Key	Type	Bounds	Options
keyAllPages ('AllP')	typeBoolean		flagsOptionalSinglePa-rameter
keyFrom('From')	typeInteger		flagsOptionalSinglePa-rameter
keyTo('T ')	typeInteger		flagsOptionalSinglePa-rameter
keyInputPDF('InpP')	typeAlias		flagsSingleParameter
keyMode ('Md ')	classType		flagsSingleParameter
keyResolution ('Rslt')	typeUnitFloat ('UntF')		flagsSingleParameter
keyAntiAlias ('AntA')	typeBoolean		flagsSingleParameter
keyBaseName ('BasN')	typeTEXT		flagsSingleParameter
keyOutputDirec-tory ('OutD')	typeAlias		flagsSingleParameter

eventResizeImage ("1333cf0c-cb67-11d1-bc43-0060b0a13dc4")

Table 11–6: eventResizeImage Parameters(4)

Key	Type	Bounds	Options
keyWidth('Wdth')	typeUnitFloat		flagsSingleParameter
keyHeight ('Hght')	typeUnitFloat		flagsSingleParameter

A. Information Sources

Much of the information about keys, structures, and terminology (especially the information not detailed in this document) can be found in the header files shipped with the Software Development Kit or SDK.

In Photoshop 5.0, there are many more header files than in Photoshop 4.0. This is because many functions are now broken out in separate files.

Generally, the header files that relate to Photoshop, actions and PICA suites are located in the Headers folder in the Sample Code folder in the SKD folder. In Headers there are two folders, Photoshop and SDK. In the Photoshop folder are PI header files and folders for ADM header files, PICA header files, and PS-Suites header files. In the SDK folder are the PIU header files.

Some of the more useful files include:

Name	Location	Description
PIUSuites.h	SampleCode/Common/Headers/SDK	Contains the twenty+ most common and useful Photoshop, ADM, and PICA suites. Defines the suite names as smart pointers and declares them as global. Photoshop suites that are not included in PIUSuites.h can be found in the PS-Suite folder in the Photoshop folder.
PIDefines.h	SampleCode/Common/Headers/SDK	Some basic definitions used for a plug-in (are you on a Mac, or PC, using Metrowerks or not, etc.)
PITypes.h	SampleCode/Common/Headers/Photoshop	Photoshop specific types. A large file that includes all of the macros defined for Photoshop.
PIGeneral.h	SampleCode/Common/Headers/Photoshop	General descriptions with more specific error dialogs, more defines, defines PiPL structures, PiPL types, the different plug-in modes, color services information, old property definitions from previous Photoshop versions, and PICA caller and selector strings.
PIUDispatch.h	SampleCode/Common/Headers/SDK	Contains the Dispatch class definitions and constructors that enable the Dispatch code in the TriggerFilters plug-in sample code. This code handles the overhead of calling PICA-style suites.
PIActions.h	SampleCode/Common/Headers/Photoshop	Contains all of the routines related to actions and scripting. In particular, it defines the PSActionControl Procedures and Suite functions and the PSActionDescriptor Procedures and Suite functions.

Name	Location	Description
PIBufferSuite.h	SampleCode/Common/Headers/Photoshop/PS-Suites	Contains type definitions for memory buffer functions and parameters.
SPBasic.h	SampleCode/Common/Headers/Photoshop/PICA	Defines basic suite structures that tell you what parameters are required to acquire a suite.
PITerminology.h	SampleCode/Common/Headers/Photoshop	All the four character keys for Photoshop classes, enumerations, events, units, forms, keys and types.

Common prefixes and conventions

SP = Sweet Pea = PICA. SP refers to the Plug-In Component Architecture (formerly known as Sweet Pea).

PS = Photoshop. PS refers to Photoshop.

PI. PI refers to Plug-Ins.

PIU = Plug-In Utilities. Utility functions and definitions created by the Developer Support group.

sXX = suite pointer . Used to indicate a reference to a suite related suite function, i.e., `sPSActionsReference->Make(&reference)` is a function call from the Photoshop Actions Suite.

B. PIUBasic Files

This diagram shows the files in the PIUBasic.mcp. These define the basic Photoshop and actions functions.

PIUBasic Files:

PIUBasic groups common functions into higher-level macros and routines in order to simplify plug-in programming.

PIUSuites.ccp Source file (C++ source file)

#includes:

PIUSuites.h

PICA Suite Definitions:

```
SPBasic_t::SPBasic_t(SPBasicSuite * inSPBasic = NULL) :
 PIUSuitePointer <SPBasicSuite>
  (
  inSPBasic,
  kSPBasicSuite,
  kSPBasicSuiteVersion
  )
 {
 }

SPRuntime_t::SPRuntime_t(SPBasicSuite * inSPBasic = NULL) :
 PIUSuitePointer <SPRuntimeSuite>
  (
  inSPBasic,
  kSPRuntimeSuite,
  kSPRuntimeSuiteVersion
  )
 {
 }
```

Action Suite Definitions:

Photoshop Memory Suites:

Photoshop User Interface and Text Suites:

Photoshop Error Management Suite:

Adobe Dialog Manager Suite References:

PIUActionParams.ccp Source file (C++ source file)

#includes:

PIUActionParams.h
PIUSuites.h

Action Suite References:

```
SPBasic_t::SPBasic_t(SPBasicSuite * inSPBasic = NULL) :
PIUActionParams_t::
 PIUActionParams_t
  (
  PIActionParameters * actionsParams = NULL
  ) :
  actionParams_(actionsParams)
 {
 }
```

PIUUIParams.ccp Source file (C++ source file)

#includes:

PIUUIParams.h

User interface parameters:

Source/Header/Library Files:

Source Files:

PIUSuites.ccp
PIUActionParams.cpp
PIUUIParams.ccp
PIUUINotifyUtils.cpp
PIUDispatch.cpp
PIUTools.cpp
PIUBasic.cpp
PIUfile.cpp

Header Files:

PIActionsParams.h
PIUDispatch.h
PIUSuites.h
PIUUIParams.h
PIUSuitePointer.h
PIUTools.h
PIUFile.h
PIUBasic.h
PIUNotifyUtils.h
PIUConstants.h

Libraries:

InterfaceLib
MSL C.PPC.Lib
MSL RuntimePPC.Lib

PIUUINotifyUtils.cpp Source file

#includes:

PIUNotifyUtils.h

```
PIUNotifyUtils_t::PIUNotifyUtils_t()
 {
 }
```

PIUDispatch.cpp Source file

#includes:

PIUDispatch.h

Functionality:

Checks who called who, acquires and releases suites, loads and reloads appropriate suites.

PIUTools.cpp Source file

#includes:

PIUTools.h
PIUSuites.h

Performs various simple string, character, number and utility (copy/match) functions.

PIUFile.cpp Source file

#includes:

PIUFile.h

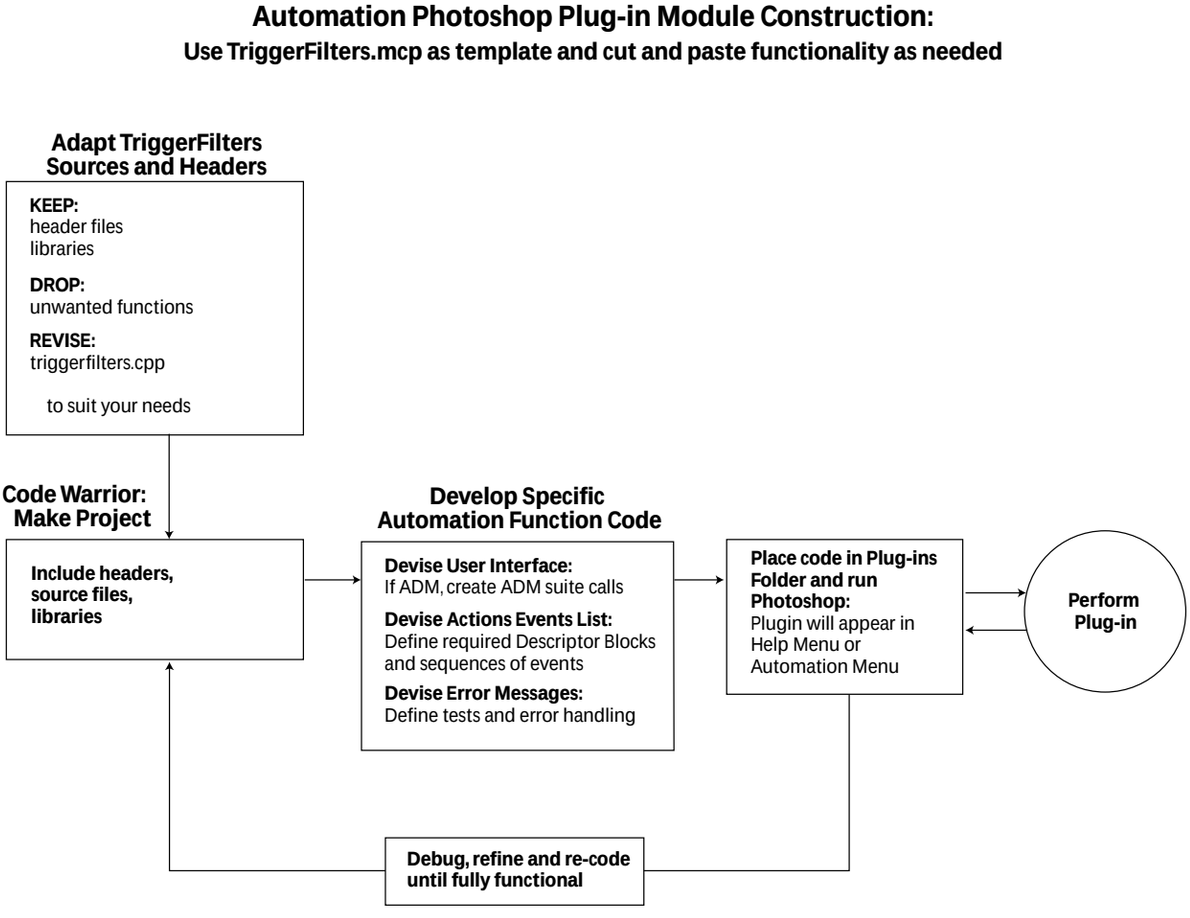
Performs various file functions.

C. Development Process

The plug-in development process follows a logical flow, and a sample automation plug-in (`TriggerFilters.8li`) is provided for you to use as a template.

First, find the sample automation plug-in closest to your needs (there may be several added as time goes by) and duplicate the entire folder including headers and code files. Then rename all of the files to your plug-in name and revise the files to suit your plug-in's needs.

The figure below illustrates this process using the Macintosh platform as a reference guide. The process using Windows as a development platform will be similar.



The fastest way to create your own Photoshop automation plug-in is to copy the project files from an existing automation plug-in and revise them to suit your needs. For example, to copy the `TriggerFilters.8li` plug-in and create a new plug-in, `MakeNew.8li`. follow the process illustrated below. The next two diagrams shows this process. Note that the finished MakeNew project is included in your SDK files as well as the TriggerFilters project.

MakeNew Plug-in Module Step-by-step Construction Part 1:

Copy TriggerFilters.mcp files and cut and paste

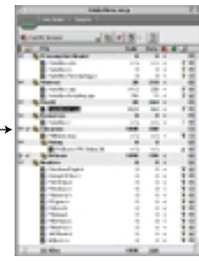
Adapt TriggerFilters Sources and Headers

KEEP:
header files
libraries

DROP:
unwanted functions

REVISE:
triggerfilters files to suit your needs

1 **Copy in Finder TriggerFilters Plug-in Project Files, and rename to MakeNew files** 2



Determine which files you'll keep

Keep all of the header and library files

3 Add newly renamed files to MakeNew project and delete the TriggerFilter files

Keep the project target settings, but change the target file name 4 to: MakeNew.8li

Revise Header Files

5 **Revise Header Info:**
Change file names in headers, etc.

Take out the TriggerFilter specific header information
Global variables, constants, prototypes, functions, etc.

Every Plug-in will need:
Execute routine, DoUI routine, Read/Write scripts

MakeNew Plug-in will need:
Variables, functions and globals for its specific functions

Revise Header Files, specifically

6 **Revise MakeNew.h file**
Change file names, global variables, constants, prototypes, functions, etc.

7 **Revise MakeNewTerminology.h file**
Change (define) the scripting key definitions

MakeNew Plug-in will need:
keyMyWidth, keyMyHeight, keyMyMode, keyImageModeType, and keyMyFill,

8 **Revise MakeNew.cpp**

Change file names, global variables, constants, prototypes, functions, etc.
1) Includes: PIDefine.h, etc.
2) Globals - unsigned shorts for W, H, Res; enumerated for Mode, etc.
3) Prototypes - entry point, initialize
4) Entrypoint - almost the same for every plugin (includes dispatch code)
5) Execute code (Read/Write ScripParams)
6) Initialize parameters routine
7) List of optional suite IDs for dispatcher code - list the suites your plugin won't need
8) Read Descriptors and determine Dialog options

Basic code:
Initializes variables, contains entrypoint main function determines dialog box options

9 **Revise MakeNewScripting.cpp**

Change file names, global variables, constants, prototypes, functions, etc.
MakeNewScripting.cpp includes:
1) Includes (same as MakeNew.cpp)
2) Prototypes
3) Read Scripting Parameters function
Revise flags, keys, types, etc.
(This acquires the descriptor block information for any action)
4) Write Scripting Parameters function
Revise flags, keys, types, etc.
(This returns the descriptor block for any action)

Scripting code:
Reads and writes Descriptor Values

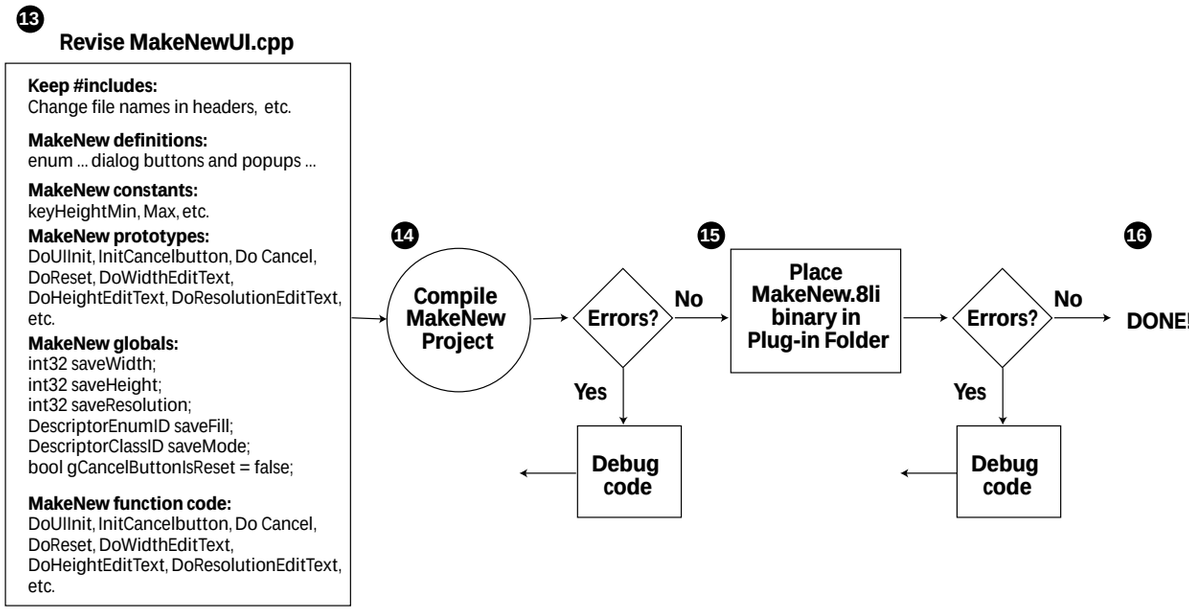
Revise User Interface via Resourcer and Derez

10 1) Read TriggerFilters.8li binary file in Resourcer and manipuate user interface elements.
2) Run Derez script in Code Warrior to turn visual resource elements into text file.
11 3) Edit text file in CW and save as MakeNew.r file. 12

User Interface Resource Editing:
Capture TriggerFilters dialog resources,
Change to suit your needs,
Save as new Resource file

MakeNew Plug-in Module Step-by-step Construction Part 2:

Copy TriggerFilters.mcp files and edit



D. Key Constants

Key Constants

The following are all the available keys supplied to you by Photoshop. They are defined in `PITerminology.h`, and are available for you to incorporate into your plug-ins. Even though it maybe necessary to create your own custom keys, it is recommended that you use as many of the built-in keys as possible.

Key Name	Constant
keyA	'A '
keyAdjustment	'Adjs'
keyAlignment	'Algn'
keyAll	'All '
keyAllExcept	'AllE'
keyAllToolOptions	'AlTl'
keyAlphaChannelOptions	'AChn'
keyAlphaChannels	'AlpC'
keyAmbientBrightness	'AmbB'
keyAmbientColor	'AmbC'
keyAmount	'Amnt'
keyAmplitudeMax	'AmMx'
keyAmplitudeMin	'AmMn'
keyAnchor	'Anch'
keyAngle	'Angl'
keyAngle1	'Ang1'
keyAngle2	'Ang2'
keyAngle3	'Ang3'
keyAngle4	'Ang4'
keyAntiAlias	'AntA'
keyAppend	'Appe'
keyApply	'Aply'
keyArea	'Ar '
keyArrowhead	'Arrw'
keyAs	'As '
keyAssumedCMYK	'AssC'
keyAssumedGray	'AssG'
keyAssumedRGB	'AssR'
keyAt	'At '
keyAuto	'Auto'
keyAutoKern	'AtKr'
keyAxis	'Axis'
keyB	'B '
keyBackground	'Bckg'

Key Name	Constant
keyBackgroundColor	'BckC'
keyBackgroundLevel	'BckL'
keyBackward	'Bwd '
keyBalance	'Blnc'
keyBeginRamp	'BgnR'
keyBeginSustain	'BgnS'
keyBevelDirection	'bvlD'
keyBevelEmboss	'ebbl'
keyBevelStyle	'bvlS'
keyBigNudgeH	'BgNH'
keyBigNudgeV	'BgNV'
keyBitDepth	'BtDp'
keyBlack	'Blck'
keyBlackClip	'BlcC'
keyBlackGeneration	'Blcn'
keyBlackGenerationCurve	'BlcG'
keyBlackIntensity	'BlcI'
keyBlackLevel	'BlcL'
keyBlackLimit	'BlcL'
keyBlendRange	'Blnd'
keyBlue	'Bl '
keyBlueBlackPoint	'BlBl'
keyBlueGamma	'BlGm'
keyBlueWhitePoint	'BlWh'
keyBlueX	'BlX '
keyBlueY	'BlY '
keyBlur	'blur'
keyBlurMethod	'BlrM'
keyBlurQuality	'BlrQ'
keyBook	'Bk '
keyBorderThickness	'BrdT'
keyBottom	'Btom'
keyBrightness	'Brgh'
keyBrushDetail	'BrsD'
keyBrushes	'Brsh'
keyBrushSize	'BrsS'
keyBrushType	'BrsT'
keyBumpAmplitude	'BmpA'
keyBumpChannel	'BmpC'
keyBy	'By '
keyByline	'Byln'
keyBylineTitle	'BylT'
keyByteOrder	'BytO'
keyCMYKSetup	'CMYS'
keyCalculation	'Clcl'
keyCaption	'Cptn'
keyCaptionWriter	'CptW'

Key Name	Constant
keyCategory	'Ctgr'
keyCellSize	'ClSz'
keyCenter	'Cntr'
keyChalkArea	'ChlA'
keyChannel	'Chnl'
keyChannelMatrix	'ChMx'
keyChannelName	'ChnN'
keyChannels	'Chns'
keyChannelsInterleaved	'ChnI'
keyCharcoalAmount	'ChAm'
keyCharcoalArea	'ChrA'
keyCity	'City'
keyClearAmount	'ClrA'
keyClippingPath	'ClPt'
keyClippingPathEPS	'ClpP'
keyClippingPathFlatness	'ClpF'
keyClippingPathIndex	'ClpI'
keyClippingPathInfo	'Clpg'
keyClosedSubpath	'Clsp'
keyColor	'Clr '
keyColorCorrection	'ClrC'
keyColorIndicates	'ClrI'
keyColorManagement	'ClMg'
keyColorPickerPrefs	'Clrr'
keyColorTable	'ClrT'
keyColorize	'Clrz'
keyColors	'Clrs'
keyColorsList	'ClrL'
keyCommandKey	'CmdK'
keyCompensation	'Cmpn'
keyCompression	'Cmpr'
keyConcavity	'Cncv'
keyConstant	'Cnst'
keyConstrain	'Cnst'
keyConstrainProportions	'CnsP'
keyContinue	'Cntn'
keyContrast	'Cntr'
keyConvert	'Cnvr'
keyCopy	'Cpy '
keyCopyright	'Cpyr'
keyCopyrightNotice	'CprN'
keyCountryName	'CntN'
keyCrackBrightness	'CrcB'
keyCrackDepth	'CrcD'
keyCrackSpacing	'CrcS'
keyCreateLayersFromLayerFX	'blfl'
keyCredit	'Crdt'

Key Name	Constant
keyCrossover	'Crss'
keyCurrentHistoryState	'CrnH'
keyCurrentLight	'CrnL'
keyCurrentToolOptions	'CrnT'
keyCurve	'Crv '
keyCurveFile	'CrvF'
keyCustom	'Cstm'
keyCyan	'Cyn '
keyDarkIntensity	'DrkI'
keyDarkness	'Drkn'
keyDateCreated	'DtCr'
keyDatum	'Dt '
keyDCS	'DCS '
keyDefinition	'Dfnt'
keyDensity	'Dnst'
keyDepth	'Dpth'
keyDestBlackMax	'Dstl'
keyDestBlackMin	'DstB'
keyDestWhiteMax	'Dstt'
keyDestWhiteMin	'DstW'
keyDetail	'Dtl '
keyDirection	'Drct'
keyDirectionBalance	'DrcB'
keyDisplaceFile	'DspF'
keyDisplacementMap	'DspM'
keyDistance	'Dstn'
keyDistortion	'Dstr'
keyDistribution	'Dstr'
keyDither	'Dthr'
keyDitherPreserve	'Dthp'
keyDitherQuality	'Dthq'
keyDocumentID	'DocI'
keyDotGain	'DtGn'
keyDotGainCurves	'DtGC'
keyDropShadow	'DrSh'
keyDuplicate	'Dplc'
keyEdge	'Edg '
keyEdgeBrightness	'EdgB'
keyEdgeFidelity	'EdgF'
keyEdgeIntensity	'EdgI'
keyEdgeSimplicity	'EdgS'
keyEdgeThickness	'EdgT'
keyEdgeWidth	'EdgW'
keyEffect	'Effc'
keyEmbedCMYK	'EmbC'
keyEmbedGray	'EmbG'
keyEmbedLab	'EmbL'

Key Name	Constant
keyEmbedRGB	'EmbR'
keyEnabled	'enab'
keyEncoding	'Encd'
keyEnd	'End '
keyEndArrowhead	'EndA'
keyEndRamp	'EndR'
keyEndSustain	'EndS'
keyEngine	'Engn'
keyExport	'Expr'.
keyExposure	'Exps'
keyExtend	'Extd'
keyExtension	'Extn'
keyExtrudeDepth	'ExtD'
keyExtrudeMaskIncomplete	'ExtM'
keyExtrudeRandom	'ExtR'
keyExtrudeSize	'ExtS'
keyExtrudeSolidFace	'ExtF'
keyExtrudeType	'ExtT'
keyEyeDropperSample	'EyDr'
keyFPXCompress	'FxCm'
keyFPXQuality	'FxQl'
keyFPXSize	'FxSz'
keyFPXView	'FxVw'
keyFeather	'Fthr'
keyFiberLength	'FbrL'
keyFileCreator	'FlCr'
keyFileInfo	'FlIn'
keyFileReference	'FilR'
keyFileType	'FlTy'
keyFill	'Fl '
keyFillColor	'FlCl'
keyFillNeutral	'FlNt'
keyFlareCenter	'FlrC'
keyFlatness	'Fltn'
keyFlatten	'Fltt'
keyFocus	'Fcs '
keyFolders	'Fldr'
keyFontName	'FntN'
keyForegroundColor	'FrgC'
keyForegroundLevel	'FrgL'
keyFormat	'Fmt '
keyForward	'Fwd '
keyFrameWidth	'FrmW'
keyFreeTransformCenterState	'FTcs'
keyFrequency	'Frqn'
keyFrom	'From'
keyFromBuiltin	'FrmB'

Key Name	Constant
keyFromMode	'FrmM'
keyFunctionKey	'FncK'
keyFuzziness	'Fzns'
keyGCR	'GCR '
keyGIFColorFileType	'GFPT'
keyGIFColorLimit	'GFCL'
keyGIFExportCaption	'GFEC'
keyGIFMaskChannelIndex	'GFMI'
keyGIFMaskChannelInverted	'GFMV'
keyGIFPaletteFile	'GFPF'
keyGIFPaletteType	'GFPL'
keyGIFRequiredColorSpaceType	'GFCS'
keyGIFRowOrderType	'GFIT'
keyGIFTransparentColor	'GFTC'
keyGIFTransparentIndexBlue	'GFTB'
keyGIFTransparentIndexGreen	'GFTG'
keyGIFTransparentIndexRed	'GFTR'
keyGIFUseBestMatch	'GFBM'
keyGamma	'Gmm '
keyGlobalAngle	'gblA'
keyGlobalLightingAngle	'gagl'
keyGloss	'Glos'
keyGlowAmount	'GlwA'
keyGrain	'Grn '
keyGrainType	'Grnt'
keyGraininess	'Grns'
keyGray	'Gry '
keyGrayBehavior	'GrBh'
keyGraySetup	'GrSt'
keyGreen	'Grn '
keyGreenBlackPoint	'GrnB'
keyGreenGamma	'GrnG'
keyGreenWhitePoint	'GrnW'
keyGreenX	'GrnX'
keyGreenY	'GrnY'
keyGridMajor	'GrdM'
keyGridMinor	'Grdn'
keyGroup	'Grup'
keyGroutWidth	'GrtW'
keyGuides	'Gdes'
keyHalftoneFile	'HlfF'
keyHalftoneScreen	'HlfS'
keyHalftoneSize	'HlSz'
keyHeader	'Hdr '
keyHeadline	'Hdln'
keyHeight	'Hght'
keyHostName	'HstN'

Key Name	Constant
keyHighlightArea	'HghA'
keyHighlightColor	'hglC'
keyHighlightLevels	'HghL'
keyHighlightMode	'hglM'
keyHighlightOpacity	'hglO'
keyHighlightStrength	'HghS'
keyHistoryBrushSource	'HstB'
keyHistoryPrefs	'HstP'
keyHistoryStateSource	'HsSS'
keyHistoryStates	'HsSt'
keyHorizontal	'Hrzn'
keyHorizontalScale	'HrzS'
keyHostVersion	'HstV'
keyHue	'H '
keyICCEngine	'ICCE'
keyICCSetupName	'ICCt'
keyImageBalance	'ImgB'
keyImport	'Impr'
keyIn	'In '
keyInherits	'c@#^'
keyInkColors	'InkC'
keyInks	'Inks'
keyInnerGlow	'IrGl'
keyInnerGlowSource	'glwS'
keyInnerShadow	'IrSh'
keyInput	'Inpt'
keyIntensity	'Intn'
keyIntent	'Inte'
keyInterfaceBevelHighlight	'IntH'
keyInterfaceBevelShadow	'Intv'
keyInterfaceBlack	'IntB'
keyInterfaceBorder	'Intd'
keyInterfaceButtonDarkShadow	'Intk'
keyInterfaceButtonDownFill	'Intt'
keyInterfaceButtonUpFill	'InBF'
keyInterfaceColorBlue2	'ICBL'
keyInterfaceColorBlue32	'ICBH'
keyInterfaceColorGreen2	'ICGL'
keyInterfaceColorGreen32	'ICGH'
keyInterfaceColorRed2	'ICRL'
keyInterfaceColorRed32	'ICRH'
keyInterfaceIconFillActive	'IntI'
keyInterfaceIconFillDimmed	'IntF'
keyInterfaceIconFillSelected	'Intc'
keyInterfaceIconFrameActive	'Intm'
keyInterfaceIconFrameDimmed	'Intr'
keyInterfaceIconFrameSelected	'IntS'

Key Name	Constant
keyInterfacePaletteFill	'IntP'
keyInterfaceRed	'IntR'
keyInterfaceWhite	'IntW'
keyInterfaceToolTipBackground	'IntT'
keyInterfaceToolTipText	'ITTT'
keyInterlace	'Intr'
keyInterlaceCreateType	'IntC'
keyInterlaceEliminateType	'IntE'
keyInterpolation	'Intr'
keyInterpolationMethod	'IntM'
keyInvert	'Invr'
keyInvertMask	'InvM'
keyInvertSource2	'InvS'
keyInvertTexture	'InvT'
keyJPEGQuality	'JPEQ'
keyKeywords	'Kywd'
keyKind	'Knd '
keyLZWCompression	'LZWC'
keyLastTransform	'LstT'
keyLayerEffects	'Lefx'
keyLayerFXVisible	'lfxv'
keyLayer	'Lyr '
keyLayerID	'LyrI'
keyLayerName	'LyrN'
keyLayers	'Lyrs'
keyLeading	'Ldng'
keyLeft	'Left'
keyLength	'Lngt'
keyLens	'Lns '
keyLevel	'Lvl '
keyLevels	'Lvls'
keyLightDark	'LgDr'
keyLightDirection	'LghD'
keyLightIntensity	'LghI'
keyLightPosition	'LghP'
keyLightSource	'LghS'
keyLightType	'LghT'
keyLightenGrout	'LghG'
keyLightness	'Lght'
keyLinkedLayerIDs	'LnkL'
keyLocalLightingAngle	'lagl'
keyLocalRange	'LclR'
keyLocation	'Lctn'
keyLog	'Log '
keyLowerCase	'LwCs'
keyLuminance	'Lmnc'
keyMagenta	'Mgnt'

Key Name	Constant
keyMakeVisible	'MkVs'
keyMapBlack	'MpBl'
keyMapping	'Mpng'
keyMaterial	'Mtrl'
keyMatrix	'Mtrx'
keyMaximum	'Mxm '
keyMaximumStates	'MxmS'
keyMerged	'Mrgd'
keyMessage	'Msge'
keyMethod	'Mthd'
keyMezzotintType	'MztT'
keyMidpoint	'Mdpn'
keyMidtoneLevels	'MdtL'
keyMinimum	'Mnm '
keyMismatchCMYK	'MsmC'
keyMismatchGray	'MsmG'
keyMismatchRGB	'MsmR'
keyMode	'Md '
keyMonochromatic	'Mnch'
keyName	'Nm '
keyNew	'Nw '
keyNonImageData	'NnIm'
keyNonLinear	'NnLn'
keyNull	typeNull
keyNumLights	'Nm L'
keyNumber	'Nmbr'
keyNumberOfChannels	'NmbO'
keyNumberOfDocuments	'NmbD'
keyNumberOfGenerators	'NmbG'
keyNumberOfLayers	'NmbL'
keyNumberOfLevels	'NmbL'
keyNumberOfPaths	'NmbP'
keyNumberOfRipples	'NmbR'
keyObjectName	'ObjN'
keyOffset	'Ofst'
keyOn	'On '
keyOpacity	'Opct'
keyOptimized	'Optm'
keyOrientation	'Ornt'
keyOriginalHeader	'OrgH'
keyOriginalTransmissionReference	'OrgT'
keyOuterGlow	'OrGl'
keyOutput	'Otpt'
keyOverprintColors	'OvrC'
keyOverrideOpen	'OvrO'
keyOverrideSave	'Ovrd'
keyPNGFilter	'PNGf'

Key Name	Constant
keyPNGInterlaceType	'PGIT'
keyPageNumber	'PgNm'
keyPagePosition	'PgPs'
keyPalette	'Plt '
keyPaletteFile	'PltF'
keyPaperBrightness	'PprB'
keyPath	'Path'
keyPathContents	'PthC'
keyPathName	'PthN'
keyPencilWidth	'Pncl'
keyPerspectiveIndex	'Prsp'
keyPhosphors	'Phsp'
keyPickerID	'PckI'
keyPickerKind	'Pckr'
keyPixelPaintSize	'PPSz'
keyPlatform	'Pltf'
keyPoints	'Pts '
keyPosition	'Pstn'
keyPosterization	'Pstr'
keyPreferBuiltin	'PrfB'
keyPreserveAdditional	'PrsA'
keyPreserveLuminosity	'PrsL'
keyPreserveTransparency	'PrsT'
keyPreferences	'Prfr'
keyPreview	'Prvw'
keyPreviewCMYK	'PrvK'
keyProfileSetup	'PrfS'
keyProvinceState	'PrvS'
keyQuality	'Qlty'
keyQuickMask	'QucM'
keyRGBSetup	'RGBS'
keyRadius	'Rds '
keyRandomSeed	'RndS'
keyRatio	'Rt '
keyRed	'Rd '
keyRedBlackPoint	'RdBl'
keyRedGamma	'RdGm'
keyRedWhitePoint	'RdWh'
keyRedX	'RdX '
keyRedY	'RdY '
keyRelative	'Rltv'
keyRelief	'Rlf '
keyResample	'Rsmp'
keyResolution	'Rslt'
keyResourceID	'RsrI'
keyResponse	'Rspn'
keyRetainHeader	'RtnH'

Key Name	Constant
keyReverse	'Rvrs'
keyRight	'Rght'
keyRippleMagnitude	'RplM'
keyRippleSize	'RplS'
keyRotate	'Rtt '
keyRulerOriginH	'RlrH'
keyRulerOriginV	'RlrV'
keyRulerUnits	'RlrU'
keySaturation	'Strt'
keySaveAndClose	'SvAn'
keySavePaths	'SvPt'
keySaving	'Svng'
keyScale	'Scl '
keyScaleHorizontal	'SclH'
keyScaleVertical	'SclV'
keyScaling	'Scln'
keyScans	'Scns'
keyScreenType	'ScrT'
keyScript	'Scrp'
keySerialString	'SrlS'
keyShadowColor	'sdwC'
keyShadowIntensity	'ShdI'
keyShadowLevels	'ShdL'
keyShadowMode	'sdwM'
keyShadowOpacity	'sdwO'
keyShape	'Shp '
keySharpness	'Shrp'
keyShearEd	'ShrE'
keyShearPoints	'ShrP'
keyShearSt	'ShrS'
keyShiftKey	'ShfK'
keySize	'Sz '
keySkew	'Skew'
keySmartBlurMode	'SmBM'
keySmartBlurQuality	'SmBQ'
keySmoothness	'Smth'
keySnapshotInitial	'SnpI'
keySoftness	'Sftn'
keySource	'Srce'
keySource2	'Src2'
keySpecialInstructions	'SpcI'
keySpherizeMode	'SphM'
keySprayRadius	'SprR'
keySquareSize	'SqrS'
keySrcBlackMax	'Srcl'
keySrcBlackMin	'SrcB'
keySrcWhiteMax	'Srcm'

Key Name	Constant
keySrcWhiteMin	'SrcW'
keyStart	'Strt'
keyStartArrowhead	'StrA'
keyState	'Stte'
keyStrength	'srgh'
keyStrength_PLUGIN	'Strg'
keyStrokeDetail	'StDt'
keyStrokeDirection	'SDir'
keyStrokeLength	'StrL'
keyStrokePressure	'StrP'
keyStrokeSize	'StrS'
keyStrokeWidth	'StrW'
keySupplementalCategories	'SplC'
keyTarget	typeNull
keyTargetPath	'Trgp'
keyTargetPathIndex	'TrgP'
keyText	'Txt '
keyTextClickPoint	'TxtC'
keyTextData	'TxtD'
keyTexture	'Txtr'
keyTextureCoverage	'TxtC'
keyTextureFile	'TxtF'
keyTextureType	'TxtT'
keyThreshold	'Thsh'
keyTileNumber	'TlNm'
keyTileOffset	'TlOf'
keyTileSize	'TlSz'
keyTitle	'Ttl '
keyTo	'T '
keyToBuiltin	'TBl '
keyToLinked	'ToLk'
keyToMode	'TMd '
keyToggleOthers	'TglO'
keyTolerance	'Tlrn'
keyTop	'Top '
keyTotalLimit	'TtlL'
keyTracking	'Trck'
keyTransferFunction	'TrnF'
keyTransparency	'Trns'
keyTransparentWhites	'TrnW'
keyTwist	'Twst'
keyType	'Type'
keyUCA	'UC '
keyURL	'URL '
keyUndefinedArea	'UndA'
keyUntitled	'Untl'
keyUpperY	'UppY'

Key Name	Constant
keyUrgency	'Urgn'
keyUseCurves	'UsCr'
keyUseGlobalAngle	'uglg'
keyUseICCProfile	'UsIC'
keyUseMask	'UsMs'
keyUserMaskEnabled	'UsrM'
keyUserMaskLinked	'Usrs'
keyUsing	'Usng'
keyValue	'Vl '
keyVector0	'Vct0'
keyVector1	'Vct1'
keyVersionFix	'VrsF'
keyVersionMajor	'VrsM'
keyVersionMinor	'VrsN'
keyVertical	'Vrtc'
keyVerticalScale	'VrtS'
keyVisible	'Vsbl'
keyWatchSuspension	'WtcS'
keyWatermark	'watr'
keyWaveType	'Wvtp'
keyWavelengthMax	'WLMx'
keyWavelengthMin	'WLMn'
keyWhiteClip	'WhtC'
keyWhiteIntensity	'WhtI'
keyWhiteIsHigh	'WhHi'
keyWhiteLevel	'WhtL'
keyWhitePoint	'WhtP'
keyWholePath	'WhPt'
keyWidth	'Wdth'
keyWindMethod	'WndM'
keyWith	'With'
keyWorkPath	'WrPt'
keyWorkPathIndex	'WrkP'
keyX	'X '
keyY	'Y '
keyYellow	'Ylw '
keyZigZagType	'ZZTy'
key_Source	keyTo

E. Event Constants

Event Constants

The following are all the available events suppiled to you by Photoshop. They are defined in `PITerminology.h`, and are available for you to incorporate into your plug-ins. Even though it maybe necessary to create your own custom events, it is recommended that you use as many of the built-in events as possible.

Event Name	Constant
eventAccentedEdges	'AccE'
eventAdd	'Add '
eventAddNoise	'AdNs'
eventAngledStrokes	'AngS'
eventBasRelief	'BsRl'
eventBatch	'Btch'
eventBlur	'Blr '
eventBlurMore	'BlrM'
eventBorder	'Brdr'
eventBrightness	'BrgC'
eventCanvasSize	'CnvS'
eventChalkCharcoal	'ChlC'
eventCharcoal	'Chrc'
eventChrome	'Chrm'
eventClose	'Cls '
eventClouds	'Clds'
eventColorBalance	'ClrB'
eventColorHalftone	'ClrH'
eventColorRange	'ClrR'
eventColoredPencil	'ClrP'
eventConteCrayon	'CntC'
eventContract	'Cntc'
eventConvertMode	'CnvM'
eventCopy	'copy'
eventCopyMerged	'CpyM'
eventCopyToLayer	'CpTL'
eventCraquelure	'Crql'
eventCrop	'Crop'
eventCrosshatch	'Crsh'
eventCrystallize	'Crst'
eventCurves	'Crvs'
eventCustom	'Cstm'
eventCut	'cut '
eventCutToLayer	'CtTL'
eventCutout	'Ct '
eventDarkStrokes	'DrkS'
eventDeInterlace	'Dntr'

Event Name	Constant
eventDefinePattern	'DfnP'
eventDefringe	'Dfrg'
eventDelete	'Dlt '
eventDesaturate	'Dstt'
eventDespeckle	'Dspc'
eventDifferenceClouds	'DfrC'
eventDiffuse	'Dfs '
eventDiffuseGlow	'DfsG'
eventDisplace	'Dspl'
eventDryBrush	'DryB'
eventDuplicate	'Dplc'
eventDustScratches	'DstS'
eventEmboss	'Embs'
eventEqualize	'Eqlz'
eventExchange	'Exch'
eventExpand	'Expn'
eventExport	'Expr'
eventExtrude	'Extr'
eventFacet	'Fct '
eventFade	'Fade'
eventFeather	'Fthr'
eventFill	'Fl '
eventFilmGrain	'FlmG'
eventFilter	'Fltr'
eventFindEdges	'FndE'
eventFlattenImage	'FltI'
eventFlip	'Flip'
eventFragment	'Frgm'
eventFresco	'Frsc'
eventGaussianBlur	'GsnB'
eventGlass	'Gls '
eventGlowingEdges	'GlwE'
eventGrain	'Grn '
eventGraphicPen	'GraP'
eventGroup	'GrpL'
eventGrow	'Grow'
eventHalftoneScreen	'HlfS'
eventHighPass	'HghP'
eventHueSaturation	'HStr'
eventImageSize	'ImgS'
eventImport	'Impr'
eventInkOutlines	'InkO'
eventIntersect	'Intr'
eventInverse	'Invs'
eventInvert	'Invr'
eventLensFlare	'LnsF'
eventLevels	'Lvls'

Event Name	Constant
eventLightingEffects	'LghE'
eventMake	'Mk '
eventMaximum	'Mxm '
eventMedian	'Mdn '
eventMergeLayers	'MrgL'
eventMergeVisible	'MrgV'
eventMezzotint	'Mztn'
eventMinimum	'Mnm '
eventMosaic	'Msc '
eventMosaicTiles	'MscT'
eventMotionBlur	'MtnB'
eventMove	'move'
eventNTSCColors	'NTSC'
eventNeonGlow	'NGlw'
eventNotePaper	'NtPr'
eventOceanRipple	'OcnR'
eventOffset	'Ofst'
eventOpen	'Opn '
eventPaintDaubs	'PntD'
eventPaletteKnife	'PltK'
eventPaste	'past'
eventPasteInto	'PstI'
eventPasteOutside	'PstO'
eventPatchwork	'Ptch'
eventPhotocopy	'Phtc'
eventPinch	'Pnch'
eventPlaster	'Plst'
eventPlasticWrap	'PlsW'
eventPlay	'Ply '
eventPointillize	'Pntl'
eventPolar	'Plr '
eventPosterEdges	'PstE'
eventPosterize	'Pstr'
eventPrint	'Prnt'
eventPurge	'Prge'
eventQuit	'quit'
eventRadialBlur	'RdlB'
eventRasterize	'Rstr'
eventRemoveBlackMatte	'RmvB'
eventRemoveLayerMask	'RmvL'
eventRemoveWhiteMatte	'RmvW'
eventReplaceColor	'RplC'
eventReset	'Rset'
eventReticulation	'Rtcl'
eventRevert	'Rvrt'
eventRipple	'Rple'
eventRotate	'Rtte'

Event Name	Constant
eventRoughPastels	'RghP'
eventSave	'save'
eventSelect	'slct'
eventSelectiveColor	'SlcC'
eventSet	'setd'
eventSharpen	'Shrp'
eventSharpenEdges	'ShrE'
eventSharpenMore	'ShrM'
eventShear	'Shr '
eventSimilar	'Smlr'
eventSmartBlur	'SmrB'
eventSmooth	'Smth'
eventSmudgeStick	'SmdS'
eventSolarize	'Slrz'
eventSpatter	'Spt '
eventSpherize	'Sphr'
eventSplitChannels	'SplC'
eventSponge	'Spng'
eventSprayedStrokes	'SprS'
eventStainedGlass	'StnG'
eventStamp	'Stmp'
eventStop	'Stop'
eventStroke	'Strk'
eventSubtract	'Sbtr'
eventSumie	'Smie'
eventTakeMergedSnapshot	'TkMr'
eventTakeSnapshot	'TkSn'
eventTextureFill	'TxtF'
eventTexturizer	'Txtz'
eventThreshold	'Thrs'
eventTiles	'Tls '
eventTornEdges	'TrnE'
eventTraceContour	'TrcC'
eventTransform	'Trnf'
eventTrap	'Trap'
eventTwirl	'Twrl'
eventUnderpainting	'Undr'
eventUndo	'undo'
eventUngroup	'Ungr'
eventUnsharpMask	'UnsM'
eventVariations	'Vrtn'
eventWaterPaper	'WtrP'
eventWatercolor	'Wtrc'
eventWave	'Wave'
eventWind	'Wnd '
eventZigZag	'ZgZg'

F. Enumerated Constants

Enumerated Constants

The following are all the available enumerations supplied by Photoshop. They are defined in `PITerminology.h`, and are available for you to incorporate into your plug-ins. Even though it maybe necessary to create your own custom enumerations, it is recommended that you use as many of the built-in enumerations as possible.

Enumeration	Constant	Type
enum1BitPerPixel	'1Bt'	typeDepth
enum72Color	'72Cl'	typeDCS
enum72Gray	'72Gr'	typeDCS
enumAdd	'Add '	typeCalculation
enumAppleRGB	'AppR'	typeBuiltinProfile
enumASCII	'ASCI'	typeEncoding
enumAskWhenOpening	'AskW'	typeProfileMismatch, typeAssumeOptions.
enumBicubic	'Bcbc'	typeInterpolation.
enumBinary	'Bnry'	typeEncoding.
enumMonitorSetup	'MntS'	typeMenuItem.
enumSRGB	'SRGB'	typeBuiltinProfile.
enum16BitsPerPixel	'16Bt'	typeDeepDepth.
enum2BitsPerPixel	'2Bts'	typeDeepDepth.
enum32BitsPerPixel	'32Bt'	typeDeepDepth.
enum4BitsPerPixel	'4Bts'	typeDeepDepth.
enum5000	'5000'	typeKelvin.
enum5500	'5500'	typeKelvin.
enum6500	'6500'	typeKelvin.
enum7500	'7500'	typeKelvin.
enum8BitsPerPixel	'EghB'	typeDeepDepth, typeDepth.
enum9300	'9300'	typeKelvin.
enumA	'A '	typeChannel.
enumAbsColorimetric	'AClr	typeIntent.
enumADSBottomst	'AdB ‘	typeAlignDistributeSelector
enumADSCentersH	'AdCH'	typeAlignDistributeSelector
enumADSCentersV	'AdCV'	typeAlignDistributeSelector

Enumeration	Constant	Type
enumADSHorizontal	'AdHr	'typeAlignDis-tributeSelector.
enumADSLefts	'AdLf'	typeAlignDistrib-uteSelector
enumADSRights	'AdRg'	typeAlignDistrib-uteSelector
enumADSTops	'AdTp	'typeAlignDis-tributeSelector.
enumADSVertical	'AdVr'	typeAlignDistrib-uteSelector.
enumAboutApp	'AbAp'	typeMenuItem. About menu.
enumAbsolute	'Absl'	typeCorrection-Method.
enumActualPixels	'ActP	'typeMenuItem. View menu.
enumAdaptive	'Adpt'	typeColorPalette.
enumAdjustmentOptions	'AdjO'	typeMenuItem. Layer menu.
enumAll	'Al '	typeOrdinal, typePurgeItem.
enumAngle	'Angl	'typeGradient-Type.
enumAny	'Any'	typeOrdinal.
enumApplyImage	'AplI'	typeMenuItem. Image menu.
enumAroundCenter	'ArnC'	typeZigZagType.
enumArrange	'Arng'	typeMenuItem. Window menu.
enumAsk	'Ask '	typeYesNo.
enumB	'B '	typeChannel.
enumBack	'Back'	typeOrdinal.
enumBackground	'Bckg'	typeFillMode. There is also a keyBackground.
enumBackgroundColor	'BckC'	typeFill, type-FillContents, typeColorStop-Type.
enumBackward	'Bckw'	typeOrdinal.
enumBehind	'Bhnd'	typeBlendMode.
enumBest	'Bst '	typeBlurQuality.
enumBetter	'Dthb'	typeDitherQual-ity.
enumBilinear	'Blnr'	typeInterpola-tion.
enumBitDepth1	'BD1 '	typeBitDepth.
enumBitDepth24	'BD24'	typeBitDepth.
enumBitDepth4	'BD4 '	typeBitDepth.
enumBitDepth8	'BD8 '	typeBitDepth.

Enumeration	Constant	Type
enumBitmap	'Btmp'	typeColorSpace.
enumBlack	'Blck'	typeGrayBehavior, typeFillContents, typeChannel.
enumBlackBody	'BlcB'	typeColorPalette.
enumBlacks	'Blks'	typeColors.
enumBlast	'Blst'	typeWindMethod.
enumBlocks	'Blks'	typeExtrudeType.
enumBlue	'Bl '	typeChannel. There is also a keyBlue.
enumBlues	'Bls '	typeColors.
enumBottom	'Bttm'	typeVerticalLocation.
enumBrushDarkRough	'BrDR'	typeBrushType.
enumBrushLightRough	'BrsL'	typeBrushType.
enumBrushSimple	'BrSm'	typeBrushType.
enumBrushSparkle	'BrSp'	typeBrushType.
enumBrushWideBlurry	'BrbW'	typeBrushType.
enumBrushWideSharp	'BrsW'	typeBrushType.
enumBuiltin	'Bltn'	typeRGBSetupSource, typeCompensation, typeCMYKSetupEngine, typeAssumeOptions.
enumButtonMode	'BtnM'	typeMenuItem. Actions palette menu.
enumCIERGB	'CRGB'	typePhosphors, typeBuiltinProfile.
enumCMYK	'CMYK'	typeChannel.
enumCMYK64	'CMSF'	typeColorSpace. CMYK Sixty-four
enumCMYKColor	'ECMY'	typeColorSpace.
enumCalculations	'Clcl'	typeMenuItem. Image menu.
enumCascade	'Cscd'	typeMenuItem. Window menu.
enumCenter	'Cntr'	typeAlignment, typeStrokeLocation.
enumCenterGlow	'SrcC'	typeInnerGlowSource.
enumChannelOptions	'ChnO'	typeMenuItem. Channels palette menu.
enumChannelsPaletteOptions	'ChnP'	typeMenuItem. Channels palette menu.
enumClear	'Clar'	typeBlendMode.

Enumeration	Constant	Type
enumClearGuides	'ClrG'	typeMenuItem. View menu.
enumClipboard	'Clpb'	typePurgeItem.
enumCloseAll	'ClsA'	typeMenuItem. Window menu.
enumCoarseDots	'CrsD'	typeMezzotint-Type.
enumColor	'Clr '	typeBlendMode.
enumColorBurn	'CBrn'	typeBlendMode, typeCalculation.
enumColorDodge	'CDdg'	typeBlendMode, typeCalculation.
enumColorMatch	'ClMt'	typeBuiltinPro-file.
enumColorimetric	'Clrm'	typeIntent.
enumComposite	'Cmps'	typeChannel.
enumConvertToCMYK	'CnvC'	typeProfileMis-match.
enumConvertToGray	'CnvG'	typeProfileMis-match.
enumConvertToLab	'CnvL'	typeProfileMis-match.
enumConvertToRGB	'CnvR'	typeProfileMis-match.
enumCreateDuplicate	'CrtD'	typeInterlaceCre-ateType.
enumCreateInterpolation	'CrtI'	typeInterlaceCre-ateType.
enumCross	'Crs '	typeShape.
enumCurrentLayer	'CrrL'	typeHistoryState-Source.
enumCustom	'Cst '	typeRGBSetup-Source. Breaks hash for enumCus-tomPattern.
enumCustomPattern	'Cstm'	typeMethod. There is also eventCus-tom, keyCustom.
enumCyan	'Cyn '	typeChannel.
enumCyans	'Cyns'	typeColors.
enumDarken	'Drkn'	typeBlendMode, typeCalculation.
enumDarkenOnly	'DrkO'	typeDiffuseMode.
enumDiamond	'Dmnd'	typeShape, type-GradientType.
enumDifference	'Dfrn'	typeBlendMode, typeCalculation.
enumDiffusion	'Dfsn'	typeDither.
enumDiffusionDither	'DfnD'	typeMethod.
enumDisplayCursorsPrefer-ences	'DspC'	typeMenuItem. File preferences menu.

Enumeration	Constant	Type
enumDissolve	'Dslv'	typeBlendMode.
enumDistort	'Dstr'	typeMenuItem. Edit transform menu.
enumDraft	'Drft'	typeBlurQuality.
enumDuotone	'Dtn '	typeChannel.
enumEBUITU	'EBT '	typePhosphors.
enumEdgeGlow	'SrcE'	typeInnerGlow-Source.
enumEliminateEvenFields	'ElmE'	typeInterlaceE-liminateType.
enumEliminateOddFields	'ElmO'	typeInterlaceE-liminateType.
enumEllipse	'Elps'	typeShape.
enumEmboss	'Embs'	typeBevelEmboss-Style. There is also an eventEm-boss.
enumExact	'Exct'	typeColorPalette.
enumExclusion	'Xclu'	typeBlendMode, typeCalculation.
enumFPXCompressLossyJPEG	'FxJP'	typeFPXCompress.
enumFPXCompressNone	'FxNo'	typeFPXCompress.
enumFaster	'Dthf'	typeDitherQual-ity.
enumFile	'Fle '	typeRGBSetup-Source.
enumFileInfo	'FlIn'	typeMenuItem. File menu.
enumFillBack	'FlBc'	typeFillColor.
enumFillFore	'FlFr'	typeFillColor.
enumFillInverse	'FlIn'	typeFillColor.
enumFillSame	'FlSm'	typeFillColor.
enumFineDots	'FnDt'	typeMezzotint-Type.
enumFirst	'Frst'	typeOrdinal.
enumFitOnScreen	'FtOn'	typeMenuItem. View menu.
enumForegroundColor	'FrgC'	typeFillCon-tents, typeColor-StopType.
enumForward	'Frwr'	typeOrdinal.
enumFreeTransform	'FrTr'	typeMenuItem. Layer menu.
enumFront	'Frnt'	typeOrdinal.
enumFullDocument	'FllD'	typeHistoryState-Source.
enumFullSize	'FlSz'	typePreview.
enumGaussianDistribution	'Gsn '	typeDistribution.
enumGIFColorFileColorTable	'GFCT'	typeGIFColorFile-Type.

Enumeration	Constant	Type
enumGIFColorFileColors	'GFCF'	typeGIFColorFile-Type.
enumGIFColorFileMicrosoft-Palette	'GFMS'	typeGIFColorFile-Type.
enumGIFPaletteAdaptive	'GFPA'	typeGIFPalette-Type.
enumGIFPaletteExact	'GFPE'	typeGIFPalette-Type.
enumGIFPaletteOther	'GFPO'	typeGIFPalette-Type.
enumGIFPaletteSystem	'GFPS'	typeGIFPalette-Type.
enumGIFRequiredColorSpace-Indexed	'GFCI'	typeGIFRequired-ColorSpaceType.
enumGIFRequiredColorSpac-eRGB	'GFRG'	typeGIFRequired-ColorSpaceType.
enumGIFRowOrderInterlaced	'GFIN'	typeGIFRowOrder-Type.
enumGIFRowOrderNormal	'GFNI'	typeGIFRowOrder-Type.
enumGeneralPreferences	'GnrP'	typeMenuItem. File preferences menu.
enumGood	'Gd '	typeBlurQuality.
enumGrainClumped	'GrnC'	typeGrainType.
enumGrainContrasty	'GrCn'	typeGrainType.
enumGrainEnlarged	'GrnE'	typeGrainType.
enumGrainHorizontal	'GrnH'	typeGrainType.
enumGrainRegular	'GrnR'	typeGrainType.
enumGrainSoft	'GrSf'	typeGrainType.
enumGrainSpeckle	'GrSp'	typeGrainType.
enumGrainSprinkles	'GrSr'	typeGrainType.
enumGrainStippled	'GrSt'	typeGrainType.
enumGrainVertical	'GrnV'	typeGrainType.
enumGrainyDots	'GrnD'	typeMezzotint-Type.
enumGraphics	'Grp '	typeIntent.
enumGray	'Gry '	typeFillCon-tents, typeChan-nel, typeColor. There is also a keyGray.
enumGray16	'GryX'	typeColorSpace. GRaY siXteen.
enumGray18	'Gr18'	typeBuiltinPro-file.
enumGray22	'Gr22'	typeBuiltinPro-file.
enumGrayScale	'Gryc'	typeColorSpace.
enumGrayscale	'Grys'	typeColorSpace, typeColorPalette.

Enumeration	Constant	Type
enumGreen	'Grn '	typeChannel, typeColor. There is also a keyGreen.
enumGreens	'Grns'	typeColors.
enumGuidesGridPreferences	'GudG'	typeMenuItem. File preferences menu.
enumHDTV	'HDTV'	typePhosphors.
enumHSBColor	'HSBl'	typeColorSpace.
enumHSLColor	'HSLC'	typeColorSpace.
enumHalftoneFile	'HlfF'	typeMethod.
enumHalftoneScreen	'HlfS'	typeMethod. There is also keyHalftoneScreen, eventHalftoneScreen.
enumHardLight	'HrdL'	typeBlendMode, typeCalculation.
enumHeavy	'Hvy '	typeBlackGeneration.
enumHideAll	'HdAl'	typeUserMaskOptions.
enumHideSelection	'HdSl'	typeUserMaskOptions.
enumHigh	'High'	typeUrgency.
enumHighQuality	'Hgh '	typeQuality.
enumHighlights	'Hghl'	typeColors.
enumHistogram	'Hstg'	typeMenuItem. Image menu.
enumHistory	'Hsty'	typePurgeItem.
enumHistoryPaletteOptions	'HstO'	typeMenuItem. History palette menu.
enumHistoryPreferences	'HstP'	typeMenuItem. File preferences menu.
enumHorizontal	'Hrzn'	typeOrientation. There is also a keyHorizontal.
enumHorizontalOnly	'HrzO'	typeSpherizeMode.
enumHue	'H '	typeBlendMode.
enumIBMPC	'IBMP'	typePlatform.
enumICC	'ICC '	typeCMYKSetupEngine, typeAssumeOptions.
enumIcon	'Icn '	typePreview.
enumIdleVM	'IdVM'	typeState
enumIgnore	'Ignr'	typeProfileMismatch.
enumImage	'Img '	typeIntent, typeAreaSelector.

Enumeration	Constant	Type
enumImageCachePreferences	'ImgP'	typeMenuItem. File preferences menu.
enumIndexedColor	'Indl'	typeColorSpace.
enumInfoPaletteOptions	'InfP'	typeMenuItem. Info palette menu.
enumInfoPaletteToggleSamplers	'InfT'	typeMenuItem. Info palette menu.
enumInnerBevel	'InrB'	typeBevelEmbossStyle.
enumInside	'Insd'	typeStrokeLocation.
enumJPEG	'JPEG'	typeDepth, typeEncoding. There is also a classJPEGFormat.
enumJustifyAll	'JstA'	typeAlignment.
enumJustifyFull	'JstF'	typeAlignment.
enumKeyboardPreferences	'KybP'	typeMenuItem. File preferences menu.
enumLab	'Lab '	typeChannel.
enumLab48	'LbCF'	typeColorSpace. LaB Color Forty-eight.
enumLabColor	'LbCl'	typeColorSpace.
enumLarge	'Lrg '	typeRippleSize.
enumLast	'Lst '	typeOrdinal.
enumLastFilter	'LstF'	typeMenuItem. Filter menu.
enumLayerOptions	'LyrO'	typeMenuItem. Layer menu.
enumLayersPaletteOptions	'LyrP'	typeMenuItem. Layers palette menu.
enumLeft	'Left'	typeHorizontalLocation.
enumLeft_PLUGIN	'Lft '	typeDirection, typeAlignment.
enumLevelBased	'LvlB'	typeExtrudeRandom.
enumLight	'Lgt '	typeBlackGeneration.
enumLightDirBottom	'LDBt'	typeLightDirection.
enumLightDirBottomLeft	'LDBL'	typeLightDirection.
enumLightDirBottomRight	'LDBR'	typeLightDirection.
enumLightDirLeft	'LDLf'	typeLightDirection.

Enumeration	Constant	Type
enumLightDirRight	'LDRg'	typeLightDirection.
enumLightDirTop	'LDTp'	typeLightDirection.
enumLightDirTopLeft	'LDTL'	typeLightDirection.
enumLightDirTopRight	'LDTR'	typeLightDirection.
enumLightDirectional	'LghD'	typeLightType.
enumLightenOnly	'LghO'	typeDiffuseMode.
enumLightOmni	'LghO'	typeLightType.
enumLightPosBottom	'LPBt'	typeLightPosition.
enumLightPosBottomLeft	'LPBL'	typeLightPosition.
enumLightPosBottomRight	'LPBr'	typeLightPosition.
enumLightPosLeft	'LPLf'	typeLightPosition.
enumLightPosRight	'LPRg'	typeLightPosition.
enumLightPosTop	'LPTp'	typeLightPosition.
enumLightPosTopLeft	'LPTL'	typeLightPosition.
enumLightPosTopRight	'LPTR'	typeLightPosition.
enumLightSpot	'LghS'	typeLightType.
enumLighten	'Lghn'	typeBlendMode, typeCalculation.
enumLightness	'Lght'	typeChannel.
enumLine	'Ln '	typeShape.
enumLinear	'Lnr '	typeGradientType.
enumLinked	'Lnkd'	typeOrdinal.
enumLongLines	'LngL'	typeMezzotintType.
enumLongStrokes	'LngS'	typeMezzotintType.
enumLow	'Low '	typeUrgency.
enumLower	'Lwr '	typeContourEdge.
enumLowQuality	'Lw '	typeQuality.
enumLuminosity	'Lmns'	typeBlendMode.
enumMacThumbnail	'McTh'	typePreview.
enumMacintosh	'Mcnt'	typePlatform, typeEPSPreview.
enumMacintoshSystem	'McnS'	typeColorPalette.
enumMagenta	'Mgnt'	typeChannel.
enumMagentas	'Mgnt'	typeColors.
enumMask	'Msk '	typeChannel.
enumMaskedAreas	'MskA'	typeMaskIndicator.

Enumeration	Constant	Type
enumMaximum	'Mxmm'	typeBlackGeneration.
enumMaximumQuality	'Mxm '	typeQuality. There is also a keyMaximum, eventMaximum.
enumMedium	'Mdim'	typeBlackGeneration, typeRippleSize.
enumMediumQuality	'Mdm '	typeQuality.
enumMediumDots	'MdmD'	typeMezzotintType.
enumMediumLines	'MdmL'	typeMezzotintType.
enumMediumStrokes	'MdmS'	typeMezzotintType.
enumMemoryPreferences	'MmrP'	typeMenuItem. File preferences menu.
enumMergeChannels	'MrgC'	typeMenuItem. Channels palette menu.
enumMerged	'Mrgd'	typeOrdinal.
enumMergedLayers	'MrgL'	typeHistoryStateSource.
enumMiddle	'Mddl'	typeOrdinal.
enumMidtones	'Mdtn'	typeColors.
enumModeGray	'MdGr'	typeMode.
enumModeRGB	'MdRG'	typeMode.
enumMonitor	'Moni'	typeRGBSetupSource, typeAssumeOptions.
enumMonotone	'Mntn'	typeChannel.
enumMulti72Color	'72CM'	typeDCS.
enumMulti72Gray	'72GM'	typeDCS.
enumMultichannel	'Mlth'	typeColorSpace.
enumMultiNoCompositePS	'NCmM'	typeDCS.
enumMultiply	'Mltp'	typeBlendMode, typeCalculation.
enumNoCompositePS	'NCmp'	typeDCS.
enumNTSC	'NTSC'	typePhosphors, typeBuiltinProfile.
enumNavigatorPaletteOptions	'NvgP'	typeMenuItem. Navigator palette menu.
enumNearestNeighbor	'Nrst'	typeInterpolation.
enumNeutrals	'Ntrl'	typeColors.
enumNewView	'NwVw'	typeMenuItem. View menu.
enumNext	'Nxt '	typeOrdinal.

Enumeration	Constant	Type
enumNikon	'Nkn '	typeLens.
enumNikon105	'Nkn1'	typeLens.
enumNo	'N '	typeYesNo.
enumNone	'None'	typeOrdinal, typePreview, typeCompensation, typeBlackGeneration, typeAssumeOptions.
enumNormal	'Nrml'	typeBlendMode, typeCalculation, typeSherizeMode, typeDiffuseMode.
enumNull		typeNull
enumOS2	'OS2 '	typePlatform.
enumOff	'Off '	typeOnOff.
enumOn	'On '	typeOnOff.
enumOpenAs	'OpAs'	typeMenuItem. File menu.
enumOrange	'Orng'	typeColor.
enumOutFromCenter	'OtFr'	typeZigZagType.
enumOutOfGamut	'OtOf'	typeColors.
enumOuterBevel	'OtrB'	typeBevelEmbossStyle.
enumOutside	'Otsd'	typeStrokeLocation.
enumOverlay	'Ovrl'	typeBlendMode, typeCalculation.
enumP22EBU	'P22B'	typePhosphors.
enumPNGFilterAdaptive	'PGAd'	typePNGFilter.
enumPNGFilterAverage	'PGAv'	typePNGFilter.
enumPNGFilterNone	'PGNo'	typePNGFilter.
enumPNGFilterPaeth	'PGPt'	typePNGFilter.
enumPNGFilterSub	'PGSb'	typePNGFilter.
enumPNGFilterUp	'PGUp'	typePNGFilter.
enumPNGInterlaceAdam7	'PGIA'	typePNGInterlaceType.
enumPNGInterlaceNone	'PGIN'	typePNGInterlaceType.
enumPagePosCentered	'PgPC'	typePagePosition.
enumPagePosTopLeft	'PgTL'	typePagePosition.
enumPageSetup	'PgSt'	typeMenuItem. File menu.
enumPalSecam	'PlSc'	typeBuiltinProfile.
enumPanaVision	'PnVs'	typeLens.
enumPathsPaletteOptions	'PthP'	typeMenuItem. Paths palette menu.

Enumeration	Constant	Type
enumPattern	'Ptrn'	typeDither, typeFillContents, typePurgeItem.
enumPatternDither	'PtnD'	typeMethod.
enumPerspective	'Prsp'	typeMenuItem. Edit transform menu.
enumPhotoshopPicker	'Phtk'	typePickerKind.
enumPillowEmboss	'PlEb'	typeBevelEmbossStyle.
enumPixelPaintSize1	'PxS1'	
enumPixelPaintSize2	'PxS2'	
enumPixelPaintSize3	'PxS3'	
enumPixelPaintSize4	'PxS4'	
enumPlace	'Plce'	typeMenuItem. File menu.
enumPlaybackOptions	'PbkO'	typeMenuItem. Actions palette menu.
enumPluginPicker	'PlgP'	typePickerKind.
enumPluginsScratchDiskPreferences	'PlgS'	typeMenuItem. File preferences menu.
enumPolarToRect	'PlrR'	typeConvert.
enumPondRipples	'PndR'	typeZigZagType.
enumPreviewOff	'PrvO'	typePreviewCMYK.
enumPreviewCMYK	'PrvC'	typePreviewCMYK.
enumPreviewCyan	'Prvy'	typePreviewCMYK.
enumPreviewMagenta	'PrvM'	typePreviewCMYK.
enumPreviewYellow	'PrvY'	typePreviewCMYK.
enumPreviewBlack	'PrvB'	typePreviewCMYK.
enumPreviewCMY	'PrvN'	typePreviewCMYK.
enumPrevious	'Prvs'	typeColorPalette, typeOrdinal.
enumPrintSize	'PrnS'	typeMenuItem. View menu.
enumPrintingInksSetup	'PrnI'	typeMenuItem.
enumPyramids	'Pyrm'	typeExtrudeType.
enumQCSAverage	'Qcsa'	typeQuadCenterState.
enumQCSCorner0	'Qcs0'	typeQuadCenterState.
enumQCSCorner1	'Qcs1'	typeQuadCenterState.
enumQCSCorner2	'Qcs2'	typeQuadCenterState.
enumQCSCorner3	'Qcs3'	typeQuadCenterState.
enumQCSIndependent	'Qcsi'	typeQuadCenterState.

Enumeration	Constant	Type
enumQCSSide0	'Qcs4'	typeQuadCenterState.
enumQCSSide1	'Qcs5'	typeQuadCenterState.
enumQCSSide2	'Qcs6'	typeQuadCenterState.
enumQCSSide3	'Qcs7'	typeQuadCenterState.
enumQuadtone	'Qdtn'	typeChannel.
enumRepeat	'Rpt '	typeFillMode.
enumRGB	'RGB '	typeGrayBehavior, typeChannel.
enumRGB48	'RGBF'	typeColorSpace. RGB Forty-eight
enumRGBColor	'RGBC'	typeColorSpace.
enumRadial	'Rdl '	typeGradientType.
enumRandom	'Rndm'	typeExtrudeRandom.
enumRectToPolar	'RctP'	typeConvert.
enumRed	'Rd '	typeChannel, typeColor.
enumRedrawComplete	'RdCm'	typeState.
enumReds	'Rds '	typeColors.
enumReflected	'Rflc'	typeGradientType.
enumRelative	'Rltv'	typeCorrectionMethod.
enumRepeatEdgePixels	'RptE'	typeUndefinedArea.
enumRevealAll	'RvlA'	typeUserMaskOptions.
enumRevealSelection	'RvlS'	typeUserMaskOptions.
enumRight	'Rght'	typeDirection, typeAlignment, typeHorizontalLocation.
enumRotate	'Rtte'	typeMenuItem. Edit transform menu.
enumRotoscopingPreferences	'RtsP'	typeMenuItem. File preferences menu.
enumRound	'Rnd '	typeShape.
enumRulerCm	'RrCm'	typeRulerUnits.
enumRulerInches	'RrIn'	typeRulerUnits.
enumRulerPercent	'RrPr'	typeRulerUnits.
enumRulerPicas	'RrPi'	typeRulerUnits.
enumRulerPixels	'RrPx'	typeRulerUnits.
enumRulerPoints	'RrPt'	typeRulerUnits.
enumSMPTE240M	'SMPT'	typePhosphors.
enumSMPTEC	'SMPC'	typePhosphors.

Enumeration	Constant	Type
enumSample3x3	'Smp3'	typeEyeDropper-Sample.
enumSample5x5	'Smp5'	typeEyeDropper-Sample.
enumSamplePoint	'SmpP'	typeEyeDropper-Sample.
enumSaturation	'Strt'	typeBlendMode.
enumSaved	'Sved'	typeFillContents.
enumSavingFilesPreferences	'SvnF'	typeMenuItem. File preferences menu.
enumScale	'Scl '	typeMenuItem. Edit transform menu. There is also a keyScale.
enumScreen	'Scrn'	typeBlendMode, typeCalculation.
enumScreenCircle	'ScrC'	typeScreenType.
enumScreenDot	'ScrD'	typeScreenType.
enumScreenLine	'ScrL'	typeScreenType.
enumSelectedAreas	'SlcA'	typeMaskIndica-tor.
enumSelection	'Slct'	typeAreaSelector.
enumSeparationSetup	'SprS'	typeMenuItem.
enumSeparationTables	'SprT'	typeMenuItem.
enumShadows	'Shdw'	typeColors.
enumShortLines	'ShrL'	typeMezzotint-Type.
enumShortStrokes	'ShSt'	typeMezzotint-Type.
enumSingle72Color	'72CS'	typeDCS.
enumSingle72Gray	'72GS'	typeDCS.
enumSingleNoCompositePS	'NCmS'	typeDCS.
enumSkew	'Skew'	typeMenuItem. Edit transform menu.
enumSmall	'Sml '	typeRippleSize.
enumSmartBlurModeEdgeOnly	'SBME'	typeSmartBlur-Mode.
enumSmartBlurModeNormal	'SBMN'	typeSmartBlur-Mode.
enumSmartBlurModeOver-layEdge	'SBMO'	typeSmartBlur-Mode.
enumSmartBlurQualityHigh	'SBQH'	typeSmart-BlurQuality.
enumSmartBlurQualityLow	'SBQL'	typeSmart-BlurQuality.
enumSmartBlurQualityMedium	'SBQM'	typeSmart-BlurQuality.
enumSnapshot	'Snps'	typeFillCon-tents, typePur-geItem.

Enumeration	Constant	Type
enumSoftLight	'SftL'	typeBlendMode, typeCalculation.
enumSpectrum	'Spct'	typeColorPalette.
enumSpin	'Spn '	typeBlurMethod.
enumSpotColor	'Spot'	typeMaskIndicator.
enumSquare	'Sqr '	typeShape.
enumStagger	'Stgr'	typeWindMethod.
enumStampIn	'In '	typeBevelEmbossStampStyle.
enumStampOut	'Out '	typeBevelEmbossStampStyle.
enumStdA	'StdA'	typeKelvin.
enumStdB	'StdB'	typeKelvin.
enumStdC	'StdC'	typeKelvin.
enumStdE	'StdE'	typeKelvin.
enumStretchToFit	'StrF'	typeDisplacementMap.
enumStrokeDirHorizontal	'SDHz'	typeStrokeDirection.
enumStrokeDirLeftDiag	'SDLD'	typeStrokeDirection.
enumStrokeDirRightDiag	'SDRD'	typeStrokeDirection.
enumStrokeDirVertical	'SDVt'	typeStrokeDirection.
enumSubtract	'Sbtr'	typeCalculation.
enumSystemPicker	'SysP'	typePickerKind.
enumTables	'Tbl '	typeCMYKSetupEngine.
enumTarget	'Trgt'	typeOrdinal.
enumTexTypeBlocks	'TxBl'	typeTextureType.
enumTexTypeBrick	'TxBr'	typeTextureType.
enumTexTypeBurlap	'TxBu'	typeTextureType.
enumTexTypeCanvas	'TxCa'	typeTextureType.
enumTexTypeFrosted	'TxFr'	typeTextureType.
enumTexTypeSandstone	'TxSt'	typeTextureType.
enumTexTypeTinyLens	'TxTL'	typeTextureType.
enumThreshold	'Thrh'	typeMethod.
enumThumbnail	'Thmb'	typePreview.
enumTIFF	'TIFF'	typeEPSPreview.
enumTile	'Tile'	typeMenuItem. Window menu.
enumTile_PLUGIN	'Tl '	typeDisplacementMap.
enumToggleActionsPalette	'TglA'	typeMenuItem. Window menu.
enumToggleBlackPreview	'TgBP'	typeMenuItem. View menu.

Enumeration	Constant	Type
enumToggleBrushesPalette	'TglB'	typeMenuItem. Window menu.
enumToggleCMYKPreview	'TglC'	typeMenuItem. View menu.
enumToggleCMYPreview	'TgCM'	typeMenuItem. View menu.
enumToggleChannelsPalette	'Tglh'	typeMenuItem. Window menu.
enumToggleColorPalette	'Tglc'	typeMenuItem. Window menu.
enumToggleCyanPreview	'TgCP'	typeMenuItem. View menu.
enumToggleEdges	'TglE'	typeMenuItem. View menu.
enumToggleGamutWarning	'TglG'	typeMenuItem. View menu.
enumToggleGrid	'TgGr'	typeMenuItem. View menu.
enumToggleGuides	'Tgld'	typeMenuItem. View menu.
enumToggleHistoryPalette	'TglH'	typeMenuItem. Window menu.
enumToggleInfoPalette	'TglI'	typeMenuItem. Window menu.
enumToggleLayerMask	'TglM'	typeMenuItem. Edit transform menu.
enumToggleLayersPalette	'Tgly'	typeMenuItem. Window menu.
enumToggleLockGuides	'TglL'	typeMenuItem. View menu.
enumToggleMagentaPreview	'TgMP'	typeMenuItem. View menu.
enumToggleNavigatorPalette	'TglN'	typeMenuItem. Window menu.
enumToggleOptionsPalette	'TglO'	typeMenuItem. Window menu.
enumTogglePaths	'TglP'	typeMenuItem. View menu.
enumTogglePathsPalette	'Tglt'	typeMenuItem. Window menu.
enumToggleRotoscopePalette	'Tglp'	typeMenuItem. Window menu.
enumToggleRulers	'TglR'	typeMenuItem. View menu.
enumToggleSnapToGrid	'TgSn'	typeMenuItem. View menu.
enumToggleSnapToGuides	'TglS'	typeMenuItem. View menu.
enumToggleStatusBar	'Tgls'	typeMenuItem. Window menu.
enumToggleSwatchesPalette	'Tglw'	typeMenuItem. Window menu.

Enumeration	Constant	Type
enumToggleToolsPalette	'TglT'	typeMenuItem. Window menu.
enumToggleYellowPreview	'TgYP'	typeMenuItem. View menu.
enumTop	'Top '	typeVerticalLocation.
enumTransparency	'Trsp'	typeChannel.
enumTransparencyGamutPreferences	'TrnG'	typeMenuItem. File preferences menu.
enumTransparent	'Trns'	typeFill.
enumTrinitron	'Trnt'	typePhosphors.
enumTritone	'Trtn'	typeChannel.
enumUndo	'Und '	typePurgeItem.
enumUniform	'Unfm'	typeColorPalette.
enumUniformDistribution	'Unfr'	typeDistribution.
enumUnitsRulersPreferences	'UntR'	typeMenuItem. File preferences menu.
enumUpper	'Upr '	typeContourEdge.
enumUserStop	'UsrS'	typeColorStopType.
enumVMPreferences	'VMPr'	typeMenuItem. File preferences menu.
enumVertical	'Vrtc'	typeOrientation.
enumVerticalOnly	'VrtO'	typeSpherizeMode.
enumViolet	'Vlt '	typeColor.
enumWaveSine	'WvSn'	typeWaveType.
enumWaveSquare	'WvSq'	typeWaveType.
enumWaveTriangle	'WvTr'	typeWaveType.
enumWeb	'Web '	typeColorPalette.
enumWhite	'Wht '	typeFill, typeFillContents.
enumWhites	'Whts'	typeColors.
enumWinThumbnail	'WnTh'	typePreview.
enumWind	'Wnd '	typeWindMethod.
enumWindows	'Win '	typePlatform.
enumWindowsSystem	'WndS'	typeColorPalette.
enumWrap	'Wrp '	typeFillMode.
enumWrapAround	'WrpA'	typeUndefinedArea.
enumYellow	'Yllw'	typeChannel.
enumYellowColor	'Ylw '	typeColor.
enumYellows	'Ylws'	typeColors.
enumYes	'Ys '	typeYesNo.
enumZip	'ZpEn'	typeEncoding.
enumZoom	'Zm '	typeLens.
enumZoomIn	'ZmIn'	typeMenuItem. View menu.

Enumeration	Constant	Type
enumZoomOut	'ZmOt'	typeMenuItem. View menu.

G. Pin Ranges

About Pin Ranges

Pin Ranges are numbers defined to restrict the values of a given variable. The variable specified is not allowed to go above the maximum value nor go below the minimum value. These numbers are internally checked by Photoshop for validity, but it would be wise if any automation plugins creating descriptors with these variables to check their validity.

Event Name	Key Name	Min	Max
Accented Edges			
	Edge Width	1	14
	Edge Brightness	0	50
	Smoothness	1	15
Add Noise			
	Amount	1	999
Angled Strokes			
	Direction Balance	0	100
	Stroke Length	3	50
	Sharpness	0	10
Bas Relief			
	Detail	1	15
	Smoothness	1	15
Brightness/Contrast			
	Brightness	-100	100
	Contrast	-100	100
Chalk & Charcoal			
	Charcoal Area	0	20
	Chalk Area	0	20
	Stroke Pressure	0	5
Charcoal			
	Charcoal Thickness	1	7
	Detail	0	5
	Light/Dark Balance	0	100
Chrome			
	Detail	0	10
	Smoothness	0	10
Color Balance			
	Cyan/Red	-100	100
	Magenta/Green	-100	100
	Yellow/Blue	-100	100
Color Halftone			
	Max Radius	4	127
	Channel 1	-360	360
	Channel 2	-360	360
	Channel 3	-360	360
	Channel 4	-360	360
Color Range			
	Fuzziness	0	200
Colored Pencil			

Event Name	Key Name	Min	Max
	Pencil Width	1	24
	Stroke Pressure	0	15
	Paper Brightness	0	50
Conte Crayon			
	Foreground Level	1	15
	Background Level	1	15
	Scaling	50	200
	Relief	0	50
Craquelure			
	Crack Spacing	2	100
	Crack Depth	0	10
	Crack Brightness	0	10
Crosshatch			
	Stroke Length	3	50
	Sharpness	0	20
	Strength	1	3
Crystallize			
	Cell Size	3	300
Cutout			
	No. Levels	2	8
	Edge Simplicity	0	10
	Edge Fidelity	1	3
Dark Strokes			
	Balance	0	10
	Black Intensity	0	10
	White Intensity	0	10
Diffuse Glow			
	Graininess	0	10
	Glow Amount	0	20
	Clear Amount	0	20
Dry Brush			
	Brush Size	0	10
	Brush Detail	0	10
	Texture	1	3
Dust & Scratches			
	Radius	1	16
	Threshold	0	255
Emboss			
	Angle	-360	360
	Height	1	10
	Amount	1	500
Extrude			
	Size	2	255
	Depth	1	255
Film Grain			
	Grain	0	20
	Highlight Area	0	20
	Intensity	0	10
Fresco			
	Brush Size	0	10
	Brush Detail	0	10
	Texture	1	3
Gaussian Blur			

Event Name	Key Name	Min	Max
	Radius	0.1	250
Glass			
	Distortion	0	20
	Smoothness	1	15
	Scaling	50	200
Glowing Edges			
	Edge Width	1	14
	Edge Brightness	0	20
	Smoothness	1	15
Grain			
	Intensity	0	100
	Contrast	0	100
Graphic Pen			
	Stroke Length	1	15
	Light/Dark Balance	0	100
Halftone Pattern			
	Size	1	12
	Contrast	0	50
High Pass			
	Radius	0.1	250
Hue/Sat			
	H	-180	180
	S	-100	100
	L	-100	100
Ink Outlines			
	Stroke Length	1	50
	Dark Intensity	0	50
	Light Intensity	0	50
Lens Flare			
	Brightness	10	300
Lighting Effects			
	Intensity	-100	100
	Focus	-100	100
	Gloss	-100	100
	Material	-100	100
	Exposure	-100	100
	Ambience	-100	100
	Height	0	100
Maximum			
	Radius	1	10
Median			
	Radius	1	16
Minimum			
	Radius	1	10
Mosaic			
	Cell Size	2	64
Mosaic Tiles			
	Tile Size	2	100
	Grout Width	1	15
	Lighten Grout	0	10
Motion Blur			
	Angle	-90	90
	Distance	1	999

Event Name	Key Name	Min	Max
Neon Glow			
	Size	-24	24
	Brightness	0	50
Note Paper			
	Image Balance	0	50
	Graininess	0	20
	Relief	0	25
Ocean Ripple			
	Ripple Size	1	15
	Ripple Magnitude	0	20
Offset			
	Horizontal	-30000	30000
	Vertical	-30000	30000
Paint Daubs			
	Brush Size	1	50
	Sharpness	0	40
Palette Knife			
	Stroke Size	1	50
	Stroke Detail	1	3
	Softness	0	10
Patchwork			
	Square Size	0	10
	Relief	0	25
Photocopy			
	Detail	1	24
	Darkness	1	50
Pinch			
	Amount	-100	100
Plaster			
	Image Balance	0	50
	Smoothness	1	15
Pointilize			
	Cell Size	3	300
Poster Edges			
	Edge Thickness	0	10
	Edge Intensity	0	10
	Posterization	0	6
Radial Blur			
	Amount	1	100
Replace Color			
	Fuzziness	0	200
	H	-180	180
	S	-100	100
	L	-100	100
Reticulation			
	Density	0	50
	Black level	0	50
	White Level	0	50
Ripple			
	Amount	-999	999
Rough Pastels			
	Stroke Length	0	40
	Stroke Detail	1	20

Event Name	Key Name	Min	Max
	Scaling	50	200
	Relief	0	50
Selective Color			
	C	-100	100
	M	-100	100
	Y	-100	100
	K	-100	100
Smart Blur			
	Radius	0.1	100
	Threshold	0.1	100
Smudge Stick			
	Length	0	10
	Highlight Area	0	20
	Intensity	0	10
Spatter			
	Spray Radius	0	25
	Smoothness	1	15
Spherize			
	Amount	-100	100
Sponge			
	Brush Size	0	10
	Definition	0	25
	Smoothness	1	15
Sprayed Strokes			
	Stroke Length	0	20
	Spray Radius	0	25
Stained Glass			
	Cell Size	2	50
	Border Thickness	1	20
	Light Intensity	0	10
Stamp			
	Light/Dark Balance	0	50
	Smoothness	1	50
Sumi-e			
	Stroke Width	3	15
	Stroke Pressure	0	15
	Contrast	0	40
Texturizer			
	Scaling	50	200
	Relief	0	50
Threshold			
	Threshold	1	255
Tiles			
	No. Tiles	1	99
	Maximum Offset	1	99
Torn Edges			
	Image Balance	0	50
	Smoothness	1	15
	Contrast	1	25
Trace Contour			
	Level	0	255
Twirl			
	Amount	-999	999

Event Name	Key Name	Min	Max
Underpainting			
	Brush Size	0	40
	Texture Coverage	0	40
	Scaling	50	200
	Relief	0	50
Unsharp Mask			
	Amount	1	500
	Radius	0.1	250
	Threshold	1	255
Water Paper			
	Fiber Length	3	50
	Brightness	0	100
	Contrast	0	100
Watercolor			
	Detail	1	14
	Intensity	0	10
	Texture	1	3
Wave			
	No. Generators	1	999
	Wavelength min	1	999
	Wavelength max	1	999
	Amplitude min	1	999
	Amplitude max	1	999
	hScale	0	100
	vScale	0	100
ZigZag			
	Amount	-100	100
	Ridges	1	20

H. Glossary

Photoshop Object Model Definitions

Action - a set of Photoshop eventswith target elements and zero or more parameters, or a single event with a target with zero or more parameters.

Class - an object or concept in Photoshop that is standardized for reuse, i.e., the Rectangle class. A Photoshop "rectangle" always consists of four numeric elements defining top, left, bottom, and right. Each Photoshop class has a class ID code and a set of defined parameters. Classes can inherit from other classes. For example, classRGBColor inherits from classColor.

Class ID - a four character key that specifies Photoshop class or object types. Spaces can be part of the key. A Class key ID is followed by a descriptor block that defines the class. Class descriptor blocks contain Property Keys as parameters.

Descriptor, Descriptor Block - an optional data structure that lists specific parameters utilized by an event. Descriptors are lists where each item consists of *key/value* combinations. If a target parameter is specified, it is always the first item in the list as the `keyTarget`. If the Target is not present, it is assumed that the event will act on the previously selected or currently selected object. Items in the Descriptor can be classes, lists, strings, integers, floating point numbers, Boolean values, aliases, enumerated values, or references to other objects.

Element - consists of objects and classes, and represents a tangible object inside Photoshop - a document, a layer, a channel, a selection, a palette, a guide, or a menu that represents an accessible component of Photoshop. Events act on elements and an element can be created, acted upon, saved in a file or destroyed. They can have simple properties, i.e., a document can be created that has a horizontal size and a vertical size. Or they can have a list of simple properties, i.e., a document can have size and a foreground color. Elements can have a list of complex properties, i.e., a user-defined selection can exist in a channel, under a layer, in a document and can have inherited properties from the color class and the mode class. Elements can be contained by other elements. A document (an element) can contain a layer (another element), that contains a channel (another element).

Enumerated Value - a specific instance or selection from a pre-defined Photoshop type. An enumerator from the Shape type which includes Round, Diamond, Ellipse, Line, Square, or Cross could be `enumRound` or `enumEllipse` or `enumLine`, etc.

Event - A Photoshop event is a basic command or instruction that can be triggered by an automation plug-in. Each Photoshop event has a 4-character ID code that acts as the name of the event and an optional Descriptor containing target information or other event-related parameters. For example, the event "Gaussian Blur" has an Event ID code of `'GsnB'`, and will be used with a Descriptor that contains the parameters for the Radius amount. Macros have been created that provide longer, more meaningful names, such as `eventGaussianBlur`. However, these macros are always evaluated to the root ID code (in this case, `'GsnB'`).

Event ID - a four character key that specifies a Photoshop event. Examples include: `eventAddNoise` (`'AdNs'`), `eventClose` (`'Cls '`), `eventFeather` (`'Fthr'`), `eventMake` (`'Mk '`), `eventSet` (`'setd'`). Note that spaces can be part of the four character ID code.

Parameter ID - a four character key that specifies the parameters or properties of a Photoshop class or event. Examples include: `keyAngle` (`'Ang1'`), `keyArea` (`'Ar '`), `keyBrushSize` (`'BrsS'`), `keyPalette` (`'Plt '`), etc. Note that spaces can be part of the key. The Parameter Key is usually followed by the specific value of the parameter.

Reference - a path through the Photoshop containment hierarchy. References can be explicit but more often are relative (to save typing errors and the like). Like a DOS file path name where "C:\harddisk\folder1\fileone" represents the explicit path name and "Folder/fileone" represents a relative path from the working directory, selection `'foo'` of "photoshop/document2/layer3/channel1" represents the explicit target, (selection 'foo') of the channel 1, of layer 3, of document 2 of application Photoshop. When evaluating a target for an event, Photoshop works its way up the containment hierarchy from the current position until it finds an element that contains a suitable target. Thus, the current position relative to the target must be factored in when selecting a target channel of "foo". If the document contains more than one layer with a channel called "foo", the target will be the first one found in the reference chain.

Type - A collection of functionally similiar values or qualities (or Enumerators) associated with a specific Photoshop event.

Type ID - a four character key that specifies what type of data is in the parameter. Photoshop supports the following data types: (typeInteger, typeFloat, typeUnitFloat, typeBoolean, typeEnumerated, and typeAlias. As you might expect, typeInteger is a whole number, typeFloat is a floating point number. The typeUnitFloat is similar to the typeFloat, but indicates that the number is a unit of value "percent" or "pixel").

Photoshop Plug-in Terminology

`hasTerminology` PiPl Property- maps Photoshop "key" information into human readable text and provides additional type information for values.

For any plug-in that contains a terminology resource, a PiPL property `hasTerminology` (`'hstm'`), must be added to the PiPL. `hasTerminology` contains the class ID, event ID, and terminology resource ID for the plug-in.

PiPL - Plug-In Property List (pronounced "pipple") is a flexible, extensible data structure for representing a plug-in module's metadata. PiPLs contain all of the information Photoshop requires to identify and load plug-in modules, as well as flags and other static properties that control the operation of each plug-in. Every plug-in will contain one or more PiPL structures. NOTE: PiPL's were introduced in Photoshop version 3.0 and replace the older Plug-in Module Information structure, or PiMI used in Photoshop versions prior to version 3.0. Under the Mac OS, PiPLs are stored as a Macintosh resource; under Windows, PiPLs are stored as Windows resources.

Properties (or property structures) - are basic units of information stored in a property list. Properties are variable length data structures, uniquely identified by a vendor code, property key, and ID number, and followed by property-specific data. Plug-in properties are stored in the Plug-in Property List or PiPL. See *Cross-application Plug-in Resource Guide* for more information.

Terminology Resource/'aete' - the definition template for mapping terminology resources. The Photoshop `'aete'` resource is the dictionary used to map text descriptors to underlying Photoshop events.

A

About Pin Ranges 220
Adjustment Inheritance 16, 17
Adjustment Inheritance Heirarchy 17
Audience 7

B

Built 133
Built-In Document Events 129
Built-in File Events 133, 166
Built-In Filter Events 85, 115

C

Classes 135
Classes and Formats 134
Color Inheritance Heirarchy 16

E

Element Action Events 120
Enumerated Constants 202
Event Constants 198

F

Filter Action Events 84
Format Inheritance Heirarchy 17
Formats 160

G

Getting Started 13, 22

K

Key Constants 179, 181, 182, 185

L

Listener 77

M

Mode Inheritance Heirarchy 16

N

No UI Mode 20

P

Pin Ranges 220
Plug-in Filter Events 96

S

Silent Mode 20

T

target chain 19
Targets 19
Targets and keyTarget 19
Tool Events 114
Types 166

U

UI Mode 20
Underlying Naming Conventions 9
Using Listener 77

V

Variable Pin Ranges 226