Photoshop CS Scripting Guide

ADOBE SYSTEMS INCORPORATED
Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000
http://partners.adobe.com

October 2003

Adobe Photoshop Scripting Guide

Table of contents

1 Introduction

1.1 About this manual

This manual provides an introduction to scripting Adobe® Photoshop® CS on Mac OS and Windows®. Chapter one covers the basic conventions used in this manual and provides an overview of requirements for scripting Photoshop.

Chapter two covers the Photoshop object model as well as generic scripting terminology, concepts and techniques. Code examples are provided in three languages:

- AppleScript
- Visual Basic
- JavaScript

Note: Separate reference manuals are available for each of these languages and accompany this Scripting Guide. The reference manuals are located on the installation CD.

Chapter three covers Photoshop-specific objects and components and describes advanced techniques for scripting the Photoshop application.

Note: Please review the **README** file shipped with Photoshop CS for late-breaking news, sample scripts, and information about oustanding issues.

1.1.1 Conventions in this guide

Code and specific language samples appear in monospaced courier font:

```
app.documents.add();
```

Several conventions are used when referring to AppleScript, Visual Basic and JavaScript. Please note the following shortcut notations:

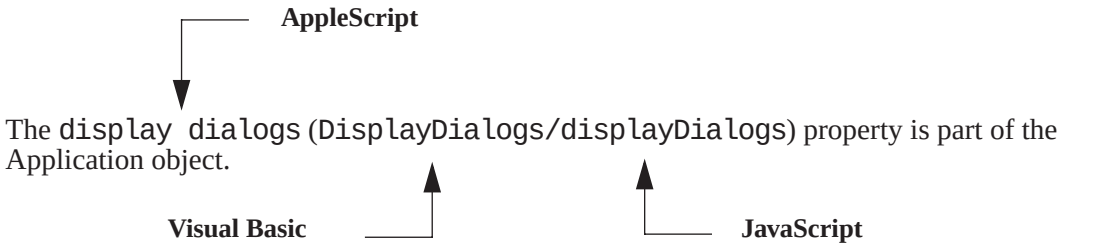
- AS stands for AppleScript
- VB stands for Visual Basic
- JS stands for JavaScript

The term “commands” will be used to refer both to commands in AppleScript and methods in Visual Basic and JavaScript.

When referring to specific properties and commands, this manual follows the AppleScript naming convention for that property and the Visual Basic and JavaScript names appear in parenthesis. For example:

“The `display dialogs` (`DisplayDialogs/displayDialogs`) property is part of the Application object.”

In this case, `display dialogs` refers to the AppleScript property, `DisplayDialogs` refers to the Visual Basic property and `displayDialogs` refers to the JavaScript property, as illustrated below.



Once you become familiar with this notation, simply focus on the values and description for the language you intend to use, ignoring the rest.

For larger blocks of code, scripting examples are often placed side by side.

AppleScript

```
layer 1 of layer set 1 of current document
```

Visual Basic

```
appRef.ActiveDocument.LayerSets(1).Layers(1)
```

JavaScript

```
app.activeDocument.layerSets[0].layers[0];
```

Finally, tables are sometimes used to organize lists of values specific to each scripting language.

1.2 What is scripting?

A script is a series of commands that tells Photoshop to perform a set of specified actions, such as applying different filters to selections in an open document. These actions can be simple, and affect only a single object, or complex and affect many objects in a Photoshop document. The actions can call Photoshop alone or invoke other applications.

Scripts automate repetitive tasks and are often used as a creative tool to streamline tasks that might be too time consuming to do manually. For example, you could write a script to generate a number of localized versions of a particular image; or to gather information about the various color profiles used by a collection of images.

1.3 Why use scripting?

Graphic design is a field characterized by creativity, but aspects of the actual work of illustration and image manipulation are anything but creative. Scripting provides a tool to help save time spent on repetitive production tasks such as resizing or reformatting documents.

Start with short, simple scripts such as those described in this manual and move on to more involved scripts as you become more proficient. Any repetitive task is a good candidate for a script. Once you can identify the steps and conditions involved in performing the task, you're ready to write a script to take care of it.

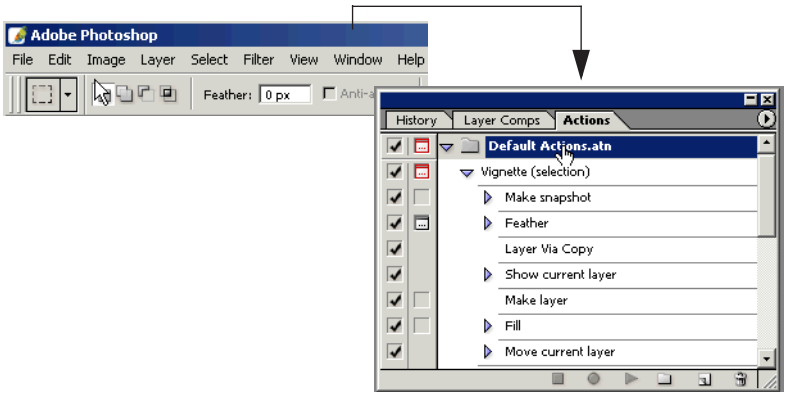
1.4 What about actions?

Photoshop actions are different from scripts. A Photoshop action is a series of tasks you have recorded while using the application—menu choices, tool choices, selection, and other commands. When you "play" an action, Photoshop performs all of the recorded commands.

Actions and scripts are two ways of automating repetitive tasks, but they work very differently.

- You cannot add conditional logic to an action. Unlike a script, actions cannot make decisions based on the current situation.
- A single script can target multiple hosts. Actions can't. For example, you could target both Photoshop and Illustrator in the same script.

The Actions palette, invoked under the Window menu, supports actions with a great deal of sophistication (including the ability to display dialogs) and allows users to work with selected objects, as illustrated below.



1.5 System requirements

The language you use to write scripts depends on your operating system: AppleScript for Mac; Visual Basic for Windows; or JavaScript, a cross-platform scripting language that can run on either Windows or Mac. While the scripting systems differ, the ways that they work with Photoshop are very similar.

1.5.1 Mac

Any system that runs Photoshop CS will support scripting. You will also need AppleScript and a script editor installed. AppleScript and the Script Editor application from Apple are included with the Mac OS. For Mac OS X, they can be found in the Applications folder. If these items are not installed on your system, reinstall them from your original system software CD-ROM.

As your scripts become more complex, you may find the need for debugging and productivity features not found in the Script Editor. There are many third-party script editors that can write and debug Apple Scripts. For more details, check:

http://www.apple.com/applescript

We use the Script Editor from Apple in this manual.

For more information on the AppleScript scripting environment, see Section 3.2.1, “Viewing Photoshop’s AppleScript dictionary

1.5.2 Windows

Any Windows system that runs Photoshop CS will support scripting. You will also need either the Windows Scripting Host, Microsoft Visual Basic, or one of the applications that contains a Visual Basic editor. Most Windows systems include the Windows Scripting Host. If you do not have Windows Scripting Host or would like more information about Windows Scripting Host, visit the Microsoft Windows Script Technologies Web site at:

(http://msdn.microsoft.com/scripting/).

We use the Microsoft Visual Basic developer framework to edit scripts in this manual.

For more information on the Visual Basic scripting environment, see Section 3.2.2, “Viewing Photoshop’s type library (VB)

1.6 JavaScript

In addition to writing AppleScripts and Visual Basic scripts, you can also write cross-platform JavaScripts (using the text editor or your choice) on either the Mac or Windows platform. Photoshop provides a built-in, platform-independent framework for executing JavaScripts.

The easiest way to run your JavaScripts is to use the “File>Scripts” menu. JavaScripts are stored in or accessed through the Scripts folder.

Scripts folder

The Scripts folder is located in the “Presets” folder of your Photoshop installation. All JavaScript files placed in the Scripts folder are available for execution from the Scripts menu.

On both Mac and Windows, JavaScript files must be saved as text files with a '.js' file name extension.

For more information on the JavaScript scripting environment, see Section 3.2.3, “Viewing the JavaScript Environment.

1.7 Choosing a scripting language

Your choice of scripting language is determined by two trade-offs:

1. Do you need to run the same script on both Macintosh and Windows computers?
2. Do you need to control multiple applications from the same script?

As mentioned earlier, JavaScript is a cross-platform language that works on either platform. The same script will perform identically on Windows and Macintosh computers. However, JavaScript is invoked from the Scripts menu within Photoshop and lacks the facilities to directly address other applications. For example, you cannot easily write a JavaScript to manage workflows involving Photoshop and a database management program.

AppleScript and Visual Basic are only offered on their respective platforms. However, you can write scripts in those languages to control multiple applications. For example, you can write an AppleScript that first manipulates a bitmap in Photoshop and then commands a web design application to incorporate it. This same cross-application capability is also available with Visual Basic on Windows.

You may also use other scripting languages when working with Photoshop. On Mac OS, any language which lets you send Apple events can be used to script Photoshop.

On Windows, any language which is COM aware can be used to script Photoshop.

1.8 Legacy COM scripting

Photoshop CS supports legacy COM scripting as long as you modify the way that you refer to the Photoshop application object in your scripts. For example, instead of saying:

```
Set appRef = CreateObject("Photoshop.Application")
```

you must change the above code to read:

```
Set appRef = CreateObject("Photoshop.Application.8.1")
```

No other change is necessary for legacy COM scripts to run under Photoshop CS.

1.9 New Features

- **Layer Comps**

  New to Photoshop CS is the ability to group layers into a "layer comp" or layer composition.

  A layer comp is a snapshot of a state of the Layers palette. Layer comps record three types of layer options: layer visibility (whether a layer in the Layers palette is showing or hidden); layer position in the document; and layer appearance (whether a layer style is applied to the layer).

  Designers often create multiple compositions or "comps" of a page layout to show clients. Using layer comps, you can create, manage and view multiple versions of a layout in a single Photoshop or ImageReady file.
- **Web Photo Gallery**

  One of the most popular features of Photoshop is the ability to create a web photo gallery out of a folder of files. You can now perform the same function through scripting.
- **JavaScripts get their own Scripts menu**

  JavaScripts now join AppleScript and Visual Basic scripts as first-class citizens. You can add your own custom JavaScripts to the Photoshop menu system. The JavaScripts you write are displayed in the “File->Scripts” menu item along with several pre-built JavaScripts that ship with the product.
- **UI for JavaScript**

  New to Photoshop CS is the ability to create graphical interface objects, such as windows and panels, employing the JavaScript programming language. UI for JavaScripts is covered in the JavaScript Reference Guide.
- **Paths**

  Although Illustrator is the premier path (or *vector*) editing application, many users want to modify their path items in Photoshop. This release of Photoshop allows you to manipulate the path items and obtain path points in a Photoshop document. Functions include the ability to create, modify, delete and copy paths using scripts.

- **PDF Presentation**

  Photoshop Elements introduced the ability to save multiple images in a single PDF document for presentation as a slide show. Photoshop CS has this feature -- called PDF Presentation -- which you can access through the scripting interface.

- **XMP Metadata**

  Metadata is any data that helps to describe the content or characteristics of a file.

  With an XMP-enabled application like Photoshop CS, information about a objects and properties and data such as camera settings and photo captions can be captured during the content creation process and embedded within the file. Meaningful descriptions and titles, searchable keywords, and up-to-date author and copyright information can be captured in a format that is easily understood by you as well as by software applications, hardware devices, and file formats.

  To access XMP metadata via scripts you also need to know XML and XAP.

1.10 What's Next

The next chapter describes the Photoshop Object Model and shows how it can be used to script "Hello, World" in three languages -- AppleScript, Visual Basic, and JavaScript. In addition, scripting basics and general scripting techniques are covered.

2 Scripting basics

2.1 Documents as objects

If you use Photoshop, then you create documents, layers, channels and design elements and probably think of a Photoshop document as a series of layers and channels — or objects. Automating Photoshop with scripting requires the same object-oriented type of thinking.

The heart of a scriptable application is the object model. In Photoshop, the object model is comprised of documents, layers and channels. Each object has its own special properties, and every object in a Photoshop document has its own identity.

This chapter covers the basic concepts of scripting within this object-oriented environment.

2.2 Object model concepts

The terminology of object-oriented programming can initially be a formidable obstacle to understanding. “Objects” belong to “classes” and have “properties” you manipulate using “commands” (AppleScript) or “methods” (Visual Basic and JavaScript). What do these words mean in this context?

It makes sense to think about objects and their properties as components of an object model. Imagine that you live in a house that responds to your commands. The house is an object, and its properties might include the number of rooms, the color of the exterior paint or the date of its construction.

Your house can also contain other objects within. Each room, for example, is an object in the house, while each window, door, or appliance is an object inside of the room. And each object can respond to various commands according to its capabilities.

Now apply this object model concept to Photoshop. The Photoshop application is the house, its documents are the rooms, and the objects in your documents are the windows and doors. You can tell Photoshop documents to add and remove objects or manipulate individual properties like color, size and shape. Actions can also be performed -- windows and doors, for example, may open and close.

2.2.1 Object classes

Objects with the same properties and behaviors are grouped into “classes.” In the house example, windows and doors belong to their own classes because they have unique properties. In Photoshop, every type of object— document, art layer, etc.—belongs to its own class, each with its own set of properties and behaviors.

2.2.2 Object inheritance

Object classes may also “inherit,” or share, the properties of a parent, or superclass. When a class inherits properties, we call that class a child, or subclass of the class from which it inherits properties. So in our house example, windows and doors are subclasses of an openings class, since they are both openings in a house. In Photoshop, art layers, for example, inherit from the layer class.

Classes often have properties that aren’t shared with their superclass. Both a window and a door, for example, inherit an "opened" property from the opening class, but a window has a "number of panes" property which the Opening class does not. In Photoshop, art layers, for example, have the property "grouped" which is not inherited from the Layer class.

2.2.3 Object elements or collections

Object elements (AppleScript) or collections (Visual Basic, JavaScript) are objects contained within other objects. For example, rooms are elements (or collections) of our house, contained within the house object. In Photoshop, documents are elements of the application object, and layers are elements of a document object. To access an element (or member of a collection), you use an index. For example, to get the first document of the application you write:

AS: `document 1`

VB: `appRef.Documents (1)`

JS: app.`documents[0];`

IMPORTANT: *Indices in AppleScript and Visual Basic are 1 based. JavaScript indices are 0 based.*

2.2.4 Object reference

The objects in Photoshop documents are arranged in a hierarchy like the house object — layers are contained within layer sets, which are placed inside a document which exists within the Photoshop application. When you send a command to a Photoshop object, you need to make sure you send the message to the right object. To do this, you identify objects by their position in the hierarchy — by an object reference. You might, for example, write the following statement.

AppleScript

```
layer 1 of layer set 1 of current document
```

Visual Basic

```
appRef.ActiveDocument.LayerSets(1).Layers(1)
```

JavaScript

```
app.activeDocument.layerSets[0].layers[0];
```

When you identify an object in this fashion, you're creating an *object reference*. While AppleScript, Visual Basic and JavaScript use different syntax for object references, each gives the script a way of finding the object you want.

2.3 Object Model

To create efficient scripts, you need to understand the containment hierarchy of the object model. Understanding the relationships among objects allows you to construct logical scripts with sound structures that contain fewer bugs and are easier to maintain.

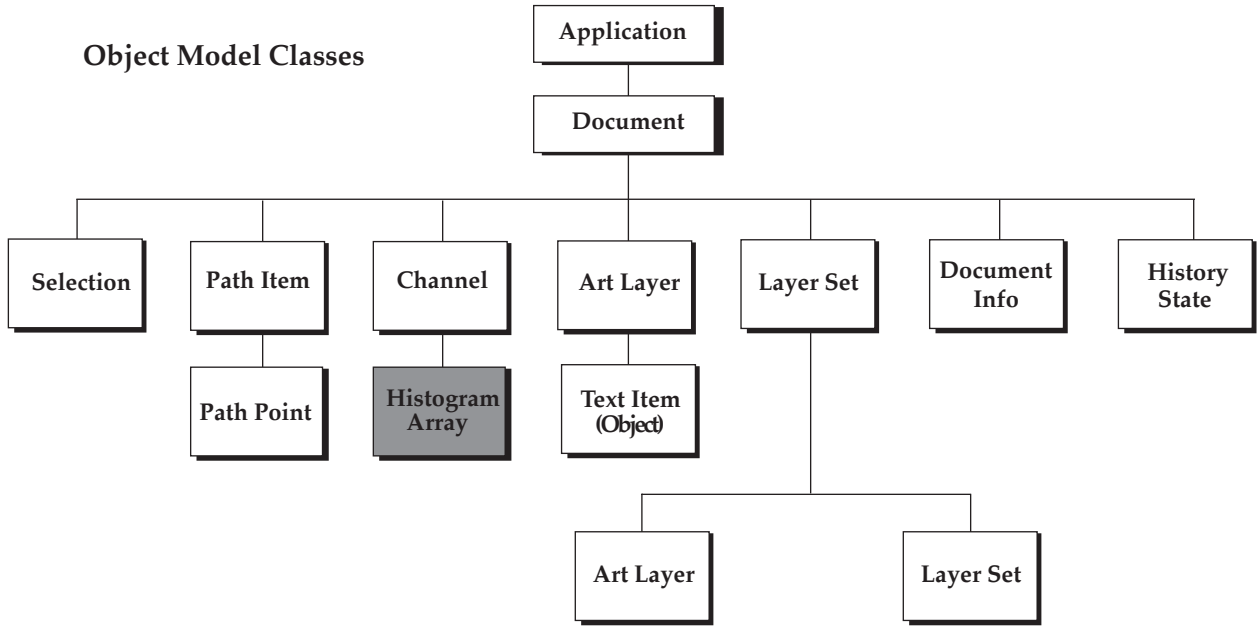
In the object model illustrated below, the Photoshop Application object sits at the top of the containment hierarchy. The Document object, directly below the Photoshop application, is the the active object you are working with and the gateway to the main components of the Photoshop object model.

Document Class

The Document class is used to make modifications to the document image. By using the `Document` object you can crop, rotate or flip the canvas, resize the image or canvas, and trim the image. You could also use the `Document` object to get the active layer, for example, save the current document, then copy and paste within the active document or between different documents.

For more information on document objects, see Section 3.9 on page 68.

.



Selection Class

The Selection class is used to specify an area of pixels in the active document (or in a selected layer of the active document) that you want to work with. For more information on selections, see Section 3.12 on page 81.

Channel Class

The Channel class is used to store pixel information about an image's color. Image color determines the number of channels available. An RGB image, for example, has four default channels: one for each primary color and one for editing the entire image. You could have the red channel active in order to manipulate just the red pixels in the image, or you could choose to manipulate all the channels at once.

These kinds of channels are related to the document mode and are called "component channels. In addition to the component channels, Photoshop lets you to create additional channels. You can create a "spot color channel", a "masked area channel" and a "selected area channel."

Using the methods of a Channel object, you can create, delete and duplicate channels. You can also retrieve a channel's histogram, change its kind or change the current channel selection. For more information on channels, see Section 3.14 on page 89.

Layer Classes

Photoshop has 2 types of layers: an `art layer` that can contain image contents and a `layer set` that can contain zero or more art layers.

An Art Layer is a layer class within a document that allows you to work on one element of an image without disturbing the others. Images are typically composed of multiple layers (see Layer Set, below). You can change the composition of an image by changing the order and attributes of the layers that comprise it.

A Text Item is a particular type of art layer that allows you to add type to an image. In Photoshop, a `text` item is implemented as a property of the art layer. For more information on text items, see Section 3.11 on page 77.

A Layer Set is a class that comprises multiple layers. Think of it as a folder on your desktop. Since folders can contain other folders, a layer set is recursive. That is, one layer set may call another layer set in the Object Model hierarchy.

For more information on layers, see Section 3.10 on page 72.

History Class

The History class is a palette object that keeps track of changes made to a document. Each time you apply a change to an image, the new state of that image is added to the palette. These states are accessible from document object and can be used to reset the document to a previous state. A history state can also be used to fill a selection. For more information on history objects, see Section 3.16 on page 94.

NOTE: In AppleScript, if you create a document and then immediately try to get history state, Photoshop returns an error. You must first activate Photoshop -- make it the front-most application -- before you can access history states.

Document Info Class

The Document Info class stores metadata about a document. Metadata is any data that helps to describe the content or characteristics of a file.For more information on document info, see Section 3.9.2 on page 70.

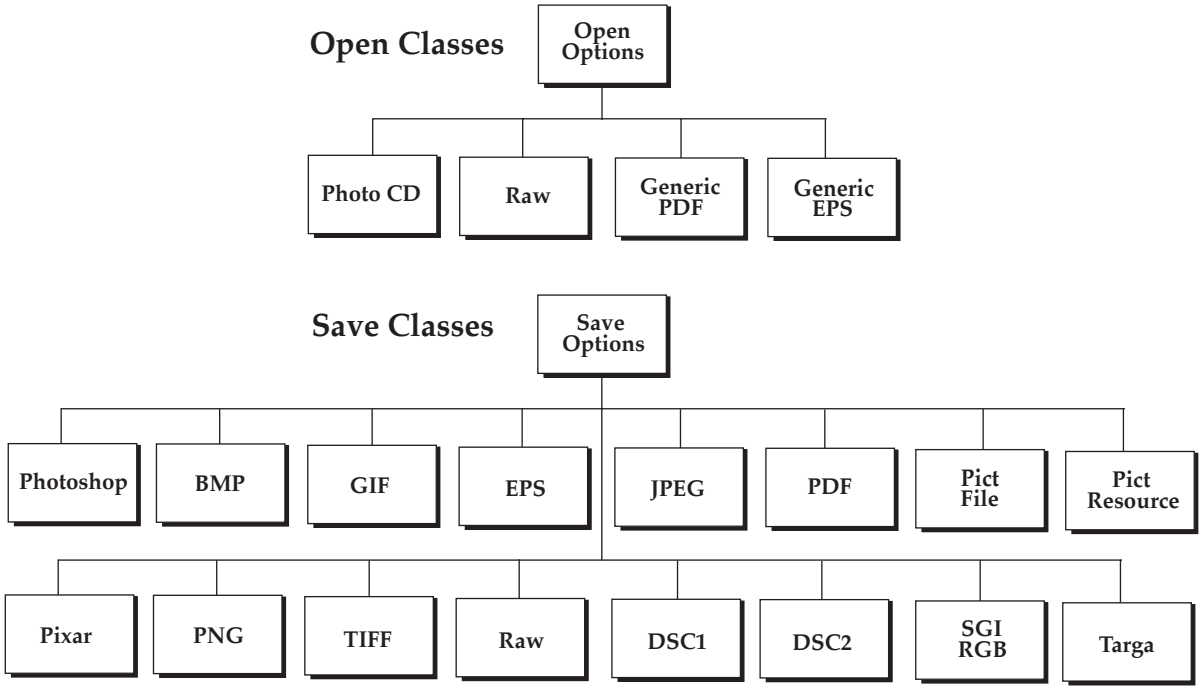
NOTE: Not shown in the Object Model are collections. A collection is a convenient way of grouping classes. Not all classes are associated with a collection.

2.3.1 Additional Containment Classes

In addition to the classes described in the Object Model, other classes allow you to open and save objects in various formats and to specify color options.

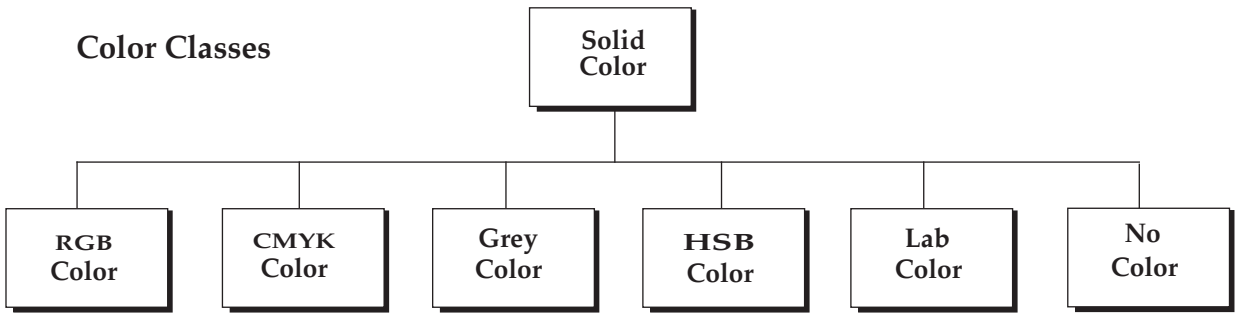
Open and Save Options

Options for opening and saving objects in Photoshop are illustrated below.



Solid Color Classes

In Visual Basic and JavaScript, the `SolidColor` object handles all colors. The solid color classes available in Photoshop are illustrated below. For more information on colors, see Section 3.15 on page 91.



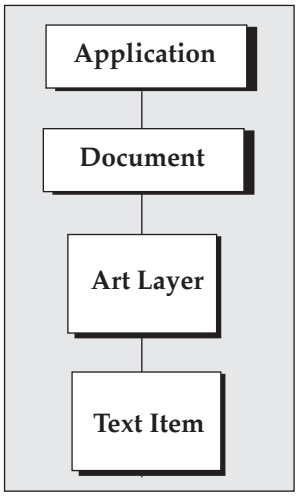
2.3.2 Hello World Sample Scripts

When all is said and done, the Object Model is simply a means to an end -- writing Photoshop scripts that accomplish something useful using the classes provided. Traditionally, the first thing to accomplish in any programming environment is the display of a "Hello World" message.

You can script such a message in Photoshop CS using AppleScript, Visual Basic or JavaScript. Regardless of the language employed, the basic steps involved are the same:

- Open the Photoshop application
- Create a new document object
- Add an art layer to the document
- Change the art layer to a text item
- Set the contents of the text item

These steps mirror a specific path in the containment hierarchy, as illustrated below.



The relationships exposed in the containment hierarchy are fully preserved in the scripting sequence. Understanding the Object Model is the key to writing effective scripts in Photoshop CS.

To see how this plays out in practice, we include three sample scripts implementing "Hello World!".

NOTE: For details on advanced scripting techniques for AppleScript, Visual Basic and JavaScript, see Chapter 3, “Scripting Photoshop.

AppleScript Sample Code

This code contains AppleScript commands that instruct the Photoshop application to create a new document with width and height in inches and place an art layer within the document. The art layer is then changed to a text layer, whose contents are set to "Hello World!".

```
tell application "Adobe Photoshop CS"

     set docRef to make new document with properties ¬
           {width:4 as inches, height:2 as inches}
     set artLayerRef to make new art layer in docRef
     set kind of artLayerRef to text layer
     set contents of text object of artLayerRef to "Hello, World!"

end tell
```

Visual Basic Sample Code

This code contains Visual Basic commands that instruct Photoshop to configure the width and height of a new document in inches and place an art layer within the document. The art layer is then changed to a text item, whose contents are set to "Hello World!". Initial unit settings are restored after the script is run.

```
Dim appRef As New Photoshop.Application

Dim originalRulerUnits As Photoshop.PsUnits
originalRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = psInches

Dim docRef As Photoshop.Document
Dim artLayerRef As Photoshop.ArtLayer
Dim textItemRef As Photoshop.TextItem
Set docRef = appRef.Documents.Add(4, 2, 72, "Hello, World!")

Set artLayerRef = docRef.ArtLayers.Add
artLayerRef.Kind = psTextLayer

Set textItemRef = artLayerRef.TextItem
textItemRef.Contents = "Hello, World!"

appRef.Preferences.RulerUnits = originalRulerUnits
```

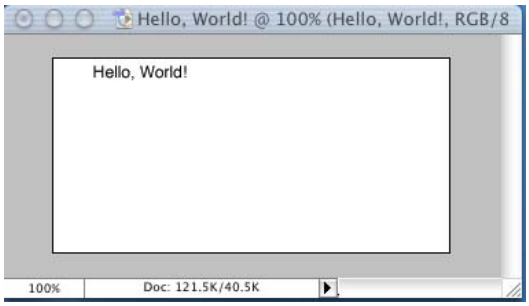
JavaScript Sample Code

This code contains JavaScript commands that instruct Photoshop to remember current unit settings and then create a new document with height and width in inches. An art layer created within the document is then changed to a text layer, whose contents are set to "Hello World!". References are released after the script is run and original ruler unit settings are restored.

```
var originalUnit = app.preferences.rulerUnits;
app.preferences.rulerUnits = Units.INCHES;

var docRef = app.documents.add( 4, 2, 72, "Hello, World!");

var artLayerRef = docRef.artLayers.add();
artLayerRef.kind = LayerKind.TEXT;

var textItemRef = artLayerRef.textItem;
textItemRef.contents = "Hello, World!";

docRef = null;
artLayerRef = null;
textItemRef = null;

app.preferences.rulerUnits = originalUnit;
```

Hello World

Aside from minor differences in display format, all three scripts produce the message illustrated below.



NOTE: The remainder of this chapter provides information about general scipting tips and techniques. Experienced AppleScript writers and Visual Basic and JavaScript programmers may want to skip to Chapter 3 for specifics on scripting Photoshop.

2.4 Documenting scripts

It's recommended that you use comments within your scripts to explain what procedures are taking place. It's a quick way to document your work for others and an important element to remember when writing scripts. Comments are ignored by the scripting system as the script executes and cause no run-time speed penalty.

AppleScript

To enter a single-line comment in an AppleScript, type "--" to the left of your description. For multiple line comments, start your comment with the characters "(*" and end it with "*)".

```
- this is a single-line comment
(* this is a
multiple line comment *)
```

Visual Basic

In Visual Basic, enter "Rem" (for "remark") or " ' " (a single straight quote) to the left of the comment.

```
Rem this is a comment
' and so is this
```

JavaScript

In JavaScript, use the double forward slash to comment a single line or a /* */ notation for multi-line comments

```
// This comments until the end of the line

/* This comments
this entire
block of text */
```

About long script lines

In some cases, individual script lines are too long to print on a single line in this guide.

AppleScript

AppleScript uses the special character (¬) to show that the line continues to the next line. This continuation character denotes a "soft return" in the script. You can enter this character in the script editor by pressing Option-Return at the end of the line you wish to continue.

Visual Basic

In Visual Basic, you can break a long statement into multiple lines in the Code window by using the line continuation character, which is a space followed by an underscore (_).

2.5 Values

Values are the data your scripts use to do their work. Most of the time, the values used in your scripts will be numbers or text.

TABLE 2.1 AppleScript Values

Value type:	What it is:	Example:
boolean	Logical true or false.	true
integer	Whole numbers (no decimal points). Integers can be positive or negative.	14
real	A number which may contain a decimal point.	13.9972
string	A series of text characters. Strings appear inside (straight) quotation marks.	"I am a string"
list	An ordered list of values. The values of a list may be any type.	{10.0, 20.0, 30.0, 40.0}
object reference	A specific reference to an object.	current document
record	An unordered list of properties, Each property is identified by its label.	{name: "you", index: 1}

TABLE 2.2 Visual Basic Values

Value type:	What it is:	Example:
Boolean	Logical true or false	True
Long	Whole numbers (no decimal points). Longs can be positive or negative.	14
Double	A number which may contain a decimal point.	13.9972
String	A series of text characters. Strings appear inside (straight) quotation marks.	"I am a string"
Array	A list of values. Arrays contain a single value type unless the type is defined as Variant.	Array(10.0, 20.0, 30.0, 40.0)
Object reference	A specific reference to an object.	appRef.ActiveDocument

TABLE 2.3 JavaScript Values

Value type:	What it is:	Example:
String	A series of text characters. Strings appear inside (straight) quotation marks.	"Hello"
Number	Any number not inside double quotes.	3.7
Boolean	Logical true or false.	true
Null	Something that points to nothing.	null
Object	Properties and methods belonging to an object or array.	activeDocument
Undefined	Devoid of any value	undefined

2.6 Variables

Variables are containers for data. A variable might contain a number, a string, a list (or array), or an object reference. Variables have names, and you refer to a variable by its name. To put data into a variable, assign the data to the variable. The file name of the current Photoshop document or the current date are both examples of data that can be assigned to a variable.

By using variables the scripts you write will be reusable in a wider variety of situations. As a script executes, it can assign data to the variables that reflect the state of the current document and selection, for example, and then make decisions based on the content of the variables.

NOTE: In AppleScript, it is not important to declare your variables before assigning values to them. In Visual Basic and JavaScript, however, it is considered good form to declare all of your variables before using them. To declare variables in Visual Basic, use the `Dim` keyword. To declare variables in JavaScript, use the `var` keyword.

2.6.1 Assigning values to variables

The remainder of this section shows how to assign values to variables.

AS

```
set thisNumber to 10
set thisString to "Hello, World!"
```

VB

```
Option Explicit
Dim thisNumber As Long
Dim thisString As String
thisNumber = 10
thisString = "Hello, World!"
```

The `Dim` statement assigns a value type to the variable, which helps keep scripts clear and readable. Memory is also used more efficiently if variables are declared before use. If you start your scripts in Visual Basic with the line `Option Explicit`, you will have to declare all variables before assigning data to them. You will not have to declare them the next time they are used.

JS

```
var x = 8;
x = x + 4;
var thisNumber = 10;
var thisString = "Hello, World!";
```

The `var` keyword identifies variables the first time that you use the variable. The next time you use the variable you should not use the var keyword.

2.6.2 Using variables to store references

Variables can also be used to store references to objects. In AppleScript, a reference is returned when you create a new object in an Photoshop document as shown below:

```
set thisLayer to make new art layer in current document
```

Or you can fill the variable with a reference to an existing object:

```
set thisLayer to art layer 1 of current document
```

Visual Basic works similarly, however, there is an important distinction to note. If you are assigning an *object reference* to a variable you must use the `Set` command. For example, to assign a variable as you create a layer, use `Set`:

```
Set thisLayer = appRef.Photoshop.ActiveDocument.ArtLayers(1)
```

or in reference to an existing layer, since it is also an *object reference*, use `Set`:

```
Set thisLayer = appRef.Photoshop.ActiveDocument.ArtLayers(1)
```

If you are trying to assign a value to a variable in Visual Basic that is not an object reference, do not use `Set`. Use Visual Basic's assignment operator, the equals sign:

```
thisNumber = 12
```

JavaScript looks similar to Visual Basic. To assign a reference to an object, you would write:

```
var docRef = app.activeDocument;
```

and to assign a value use the following:

```
var thisNumber = 12
```

2.6.3 Naming variables

It's a good idea to use descriptive names for your variables—something like `firstPage` or `corporateLogo`, rather than `x` or `c`. You can also give your variable names a standard prefix so that they'll stand out from the objects, commands, and keywords of your scripting system.

Variable names must be a single word, but you can use internal capitalization (such as `myFirstPage`) or underscore characters (`my_first_page`) to create more readable names. Variable names cannot begin with a number, and they can't contain punctuation or quotation marks.

2.7 Operators

Operators perform calculations (addition, subtraction, multiplication, and division) on variables or values and return a result. For example:

```
docWidth/2
```

would return a value equal to half of the content of the variable `docWidth`. So if `docWidth` contained the number `20.5`, the value returned would be `10.25`.

You can also use operators to perform comparisons (equal to, not equal to, greater than, or less than, etc.). Some operators differ between AppleScript, Visual Basic and JavaScript. Consult your scripting language for operators that may be unique to your OS.

AppleScript and Visual Basic use the ampersand (&) as the concatenation operator to join two strings.

```
"Pride " & "and Prejudice."
```

would return the string “Pride and Prejudice.”

JavaScript uses the “+” operator to concatenate strings.

"Pride" + " and Prejudice"

would return the string “Pride and Prejudice.”

2.8 Commands and methods

Commands (AppleScript) or methods (Visual Basic and JavaScript) are what makes things happen in a script. The type of the object you’re working with determines how you manipulate it.

AS

In AppleScript, use the `make` command to create new objects, the `set` command to assign object references to variables and to change object properties, and the `get` command to retrieve objects and their properties.

VB

In Visual Basic, use the Add method to create new objects, the Set statement to assign object references to Visual Basic variables or properties and the assignment operator (=) to retrieve and change object properties.

JS

In JavaScript, use the `add()` method to create new objects, and the assignment operator (=) to assign both object references and variables.

2.8.1 Conditional statements

Conditional statements make decisions — they give your scripts a way to evaluate something like the blend mode of a layer or the name or date of a history state — and then act according to the result. Most conditional statements start with the word `if` in all three scripting systems.

The following examples check the number of currently open documents. If no documents are open, the scripts display a messages in a dialog box.

AS

```
tell application "Adobe Photoshop CS"
     set documentCount to count every document
     if documentCount = 0 then
        display dialog "No Photoshop documents are open!"
     end if
end tell
```

VB

```
Private Sub Command1_Click()
     Dim documentCount As long
     Dim appRef As New Photoshop.Application
     documentCount = appRef.Documents.Count
     If documentCount = 0 Then
        MsgBox "No Photoshop documents are open!"
     End If
End Sub
```

JS

```
var documentCount = documents.length;
if (documentCount == 0)
{
     alert("There are no Photoshop documents open");
}
```

2.8.2 Control structures

Control structures provide for repetitive processes, or "loops." The idea of a loop is to repeat some action, with or without changes each time through the loop, until a condition is met.

Both AppleScript and Visual Basic have a variety of different control structures to choose from. The simplest form of a loop is one that repeats a series of script operations a set number of times.

AS

```
repeat with counter from 1 to 3
    display dialog counter
end repeat
```

VB

```
For counter = 1 to 3
    MsgBox counter
Next
```

JS

```
for (i = 1; i < 4; ++i)
{
    alert(i);
}
```

A more complicated type of control structure includes conditional logic, so that it loops while or until some condition is true or false.

AS

```
set flag to false
repeat until flag = true
    set flag to button returned of (display dialog "Quit?" ¬
        buttons {"Yes", "No"}) = "Yes"
end repeat

set flag to false
repeat while flag = false
    set flag to button returned of (display dialog "Later?" ¬
       buttons {"Yes", "No"}) = "No"
end repeat
```

VB

```
flag = False
Do While flag = False
    retVal = MsgBox("Quit?", vbOKCancel)
If (retVal = vbCancel) Then
  flag = True
End If
Loop

flag = False
Do Until flag = True
    retVal = MsgBox("Quit?", vbOKCancel)
    If (retVal = vbOK) Then
        flag = True
    End If
Loop
```

JS

```
var flag = false;
while (flag == false)
{
    flag = confirm("Are you sure?");
}

var flag = false;
do
{
    flag = confirm("Are you sure?");
}
while (flag == false);
```

2.9 Handlers, subroutines and functions

Subroutines, or handlers (in AppleScript) and functions (in JavaScript), are scripting modules you can refer to from within your script. These subroutines provide a way to re-use parts of scripts. Typically, you send one or more values to a subroutine and it returns one or more values.

There's nothing special about the code used in subroutines — they are conveniences that save you from having to retype the same code lines in your script.

AS

```
set flag to DoConfirm("Are you sure?")
display dialog flag as string

on DoConfirm(prompt)
     set button to button returned of (display dialog prompt ¬
        buttons {"Yes", "No"} default button 1)
     return button = "Yes"
end DoConfirm
```

VB

```
Private Sub ScriptSample_Click(Index As Integer)
    result = DoConfirm("Are you sure?")
    MsgBox (result)
End Sub

Function DoConfirm(prompt) As Boolean
    buttonPressed = MsgBox(prompt, vbYesNo)
    DoConfirm = (buttonPressed = vbYes)
End Function
```

JS

```
var theResult = DoConfirm( "Are you sure?" );

alert(theResult);

function DoConfirm(message)
{
     var result = confirm(message);
     return result;
}
```

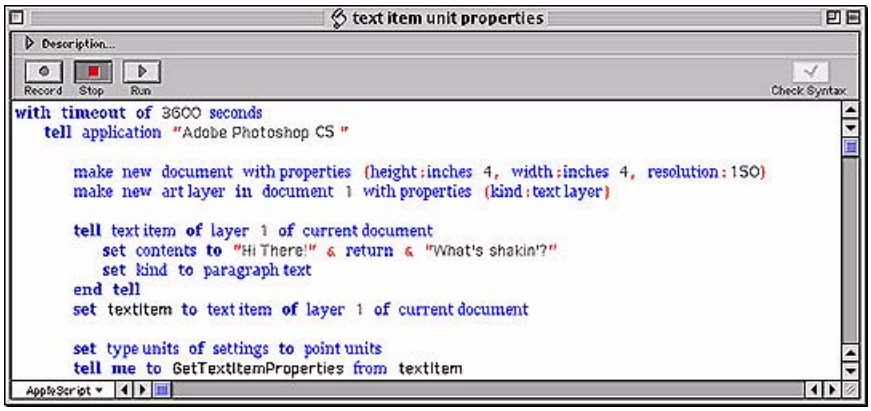
2.10 Debugging and Error Handling

Scripting environments provide tools for monitoring the progress of your script while it is running, which make it easier for you to track down any problems your script might be encountering or causing.

2.10.1 AppleScript debugging

While the basic syntax of your script will be checked when compiled, it is possible to create and compile scripts in AppleScript that will not run properly. The Script Editor Application doesn't have extensive debugging tools, but it does have the an Event Log window.

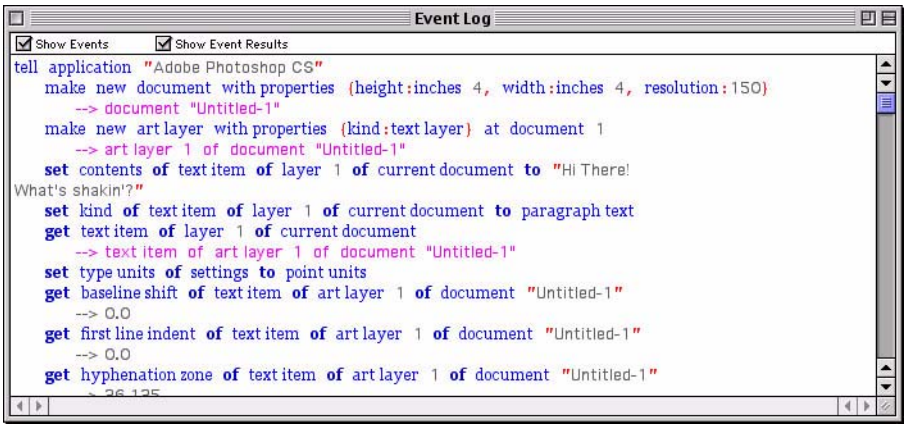
To watch the commands your script sends and the results it receives, choose "Controls > Open Event Log". The Script Editor displays the Event Log window. Check the "Show Events" and "Show Events Results" options at the top of the "Event Log" window and run your script. As the script executes, you'll see the commands sent to Photoshop, and the Photoshop responses.



You can display the contents of one or more variables in the log window by using the log command.

```
log {myVariable, otherVariable}
```

In addition, the Result window (choose Controls > Show Result) will display the value from the last script statement evaluated. Third-party editors offer additional debugging features.



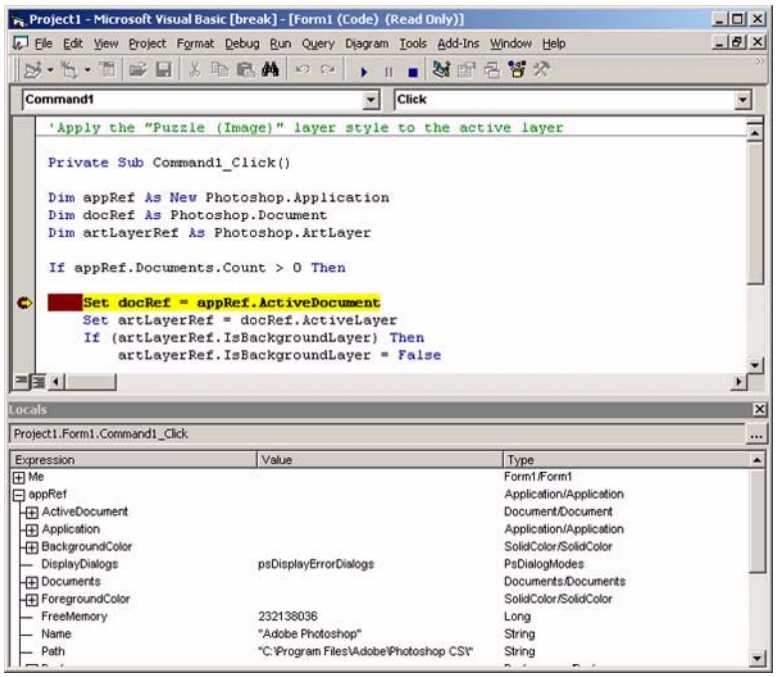
For more information on using AppleScript, see the AppleScript Reference Guide on the installation CD.

2.10.2 Visual Basic debugging

In Visual Basic, you can stop your script at any point, or step through your script one line at a time. To stop your script at a particular line, select that line in your script and choose “Debug > Toggle Breakpoint”.

When you run the script, Visual Basic will stop at the breakpoint you have set. Choose “Debug > Step Into” (or press F8) to execute the next line of your script, or choose “Run > Start” (or press F5) to continue normal execution of the script.

You can also observe the values of variables defined in your script using the “Watch” window — a very valuable tool for debugging your scripts. To view a variable in the “Watch” window, select the variable and choose “Debug > Quick Watch”. Visual Basic displays the “Quick Watch” dialog box. Click the “Add” button. Visual Basic displays the “Watch” window. If you have closed the "Watch" window, you can display it again by choosing “View > Watch Window.”



Check your Visual Basic documentation for more information. Windows Scripting Host also provides debugging information. For more information on using Visual Basic with Photoshop, see the VisualBasic Reference Guide on the installation CD.

2.10.3 JavaScript Debugging

JavaScript debugging is described in detail in the JavaScript Reference Guide on the Photoshop installation CD. Please refer to that document for further information.

2.10.4 Error handling

The following examples show how to stop a script from executing when a specific file cannot be found.

AS

```
--Store a reference to the document with the name "My Document"
--If it does not exist, display an error message
tell application "Adobe Photoshop CS"
    try
        set docRef to document "My Document"
        display dialog "Found 'My Document' "

    on error
        display dialog "Couldn't locate document 'My Document'"
    end try
end tell
```

VB

```
Private Sub Command1_Click()
' Store a reference to the document with the name "My Document"
' If the document does not exist, display an error message.
    Dim appRef As New Photoshop.Application
    Dim docRef As Photoshop.Document
    Dim errorMessage As String
    Dim docName As String

    docName = "My Document"
    Set docRef = appRef.ActiveDocument
    On Error GoTo DisplayError
        Set docRef = appRef.Documents(docName)
        MsgBox "Document Found!"
    Exit Sub
DisplayError:
    errorMessage = "Couldn't locate document " & "'" & docName & "'"
    MsgBox errorMessage
End Sub
```

JS

```
try
{
     for (i = 0; i < app.documents.length; ++i)
     {
        var myName = app.documents[i].name;
        alert(myName);
     }
}
catch(someError)
{
     alert( "JavaScript error occurred. Message = " +
           someError.description);
}
```

2.11 What's Next

The next chapter covers Photoshop-specific objects and components and describes advanced techniques for scripting the Photoshop application. The "Hello, World!" example is entended to include text filtering and document selection.

3 Scripting Photoshop

3.1 Photoshop scripting guidelines

Once you are used to thinking of Photoshop as an object oriented environment, as discussed in Chapter two, you are ready to move on to writing scripts for the application.

The following guidelines will help save debugging time when running Photoshop scripts.

- Before running scripts make sure Photoshop's Text Tool is not selected and no dialog boxes are displayed to avoid script run-time errors.
- Select documents by name rather than numeric index and set the current document in your script before working on it. Document numbers do not represent their stacking order. See 3.5, "Object references" on page 55 for more information.
- In AppleScript always create your document with a name and later get that document by name.

```
-- get the front-most document
set docRef to make new document with properties ¬
     { height:pixels 144, width:pixels 144, resolution:50,¬
          name:"My Document"}
```

- When working in VB or JavaScript, store the document reference to a newly-created document to reuse later.
- When running AppleScripts and two documents are open with the same name, both documents will be modified when the name is referenced. For example, the following script would modify the color profile of all open documents named "MyDocument."

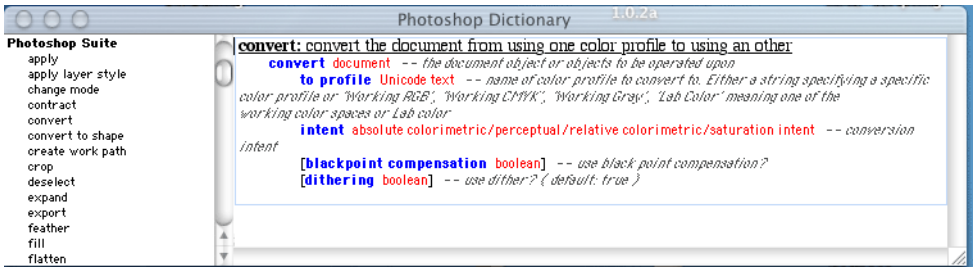
```
tell application "Adobe Photoshop CS"
     set color profile kind of document "MyDocument" to none
end tell
```

3.2 Viewing Photoshop objects, commands and methods

This section shows how to view Photoshop's objects, commands and properties in AppleScript and Visual Basic editors. JavaScript is a cross-platform language and therefore does not include a native script editor.

3.2.1 Viewing Photoshop's AppleScript dictionary

1. Start Photoshop, then your "Script Editor."
2. In Script Editor, choose "File > Open Dictionary". Script Editor displays an "Open Dictionary" dialog.
3. Find and select the Photoshop application and click the "Open" button. Script Editor displays a list of Photoshop's objects and commands and the properties and elements associated with each object, as well as the parameters for each command.



NOTE: When viewing the Photoshop dictionary using Apple's Script Editor, the complete list of open and save formats cannot be displayed because of the large number of choices available. The complete list of available open and save formats are listed below.

open **anything** -- the file(s) to be opened

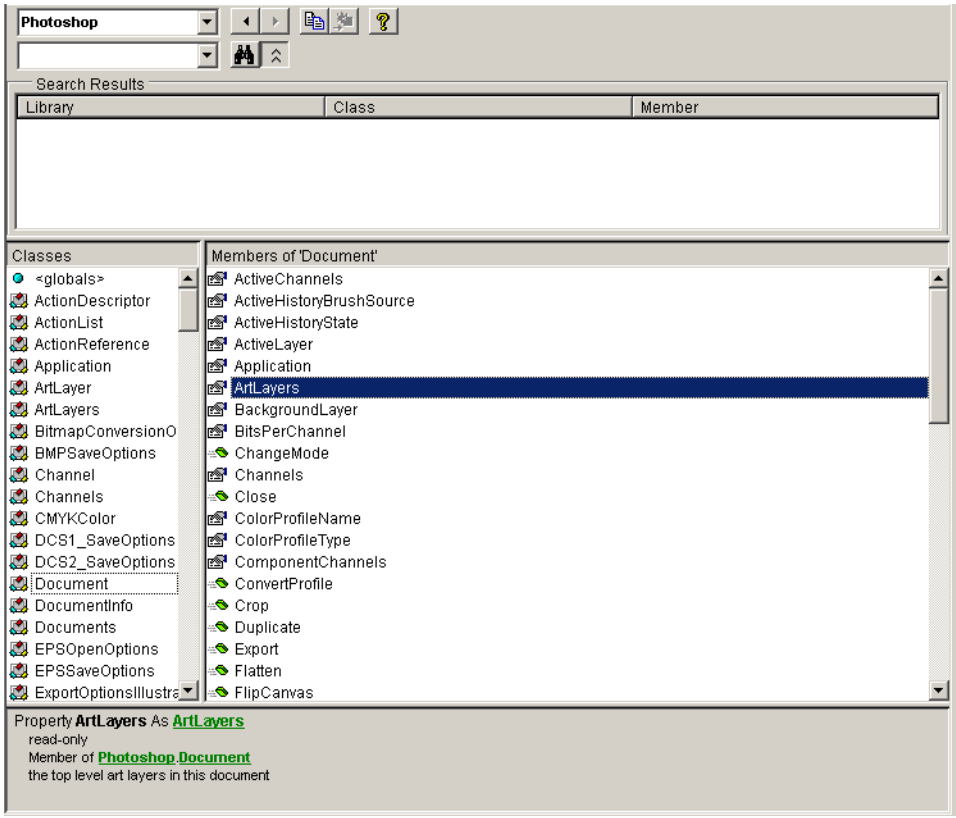
[as Acrobat TouchUp Image/Alias PIX/BMP/CompuServe GIF/ EPS/EPS PICT preview/EPS TIFF Preview/Electric Image/Filmstrip/JPEG/ PCX/PDF/PICT file/PICT resource/PNG/Photo CD/Photoshop DCS 1.0/ Photoshop DCS 2.0/Photoshop EPS/Photoshop format/Photoshop PDF/ Pixar/Portable Bitmap/raw/SGI RGB/Scitex CT/SoftImage/TIFF/Targa/ Wavefront RLA/Wireless Bitmap]

save **reference** -- the object or objects to be operated upon

[as Alias PIX/BMP/CompuServe GIF/Electric Image/JPEG/PCX/ PICT file/PICT resource/PNG/Photoshop DCS 1.0/Photoshop DCS 2.0/ Photoshop EPS/Photoshop PDF/Photoshop format/Pixar/Portable Bitmap/ raw/SGI RGB/Scitex CT/SoftImage/TIFF/Targa/Wavefront RLA/Wireless Bitmap]

3.2.2 Viewing Photoshop's type library (VB)

1. In any Visual Basic project, choose "Project > References." If you are using a built-in editor in a VBA application, choose "Tools > References."
2. Turn on the "Adobe Photoshop CS Object Library" option from the list of available references and click the "OK" button.
3. Choose "View > Object Browser." Visual Basic displays the "Object Browser" window.
4. Choose "Photoshop" from the list of open libraries shown in the top-left pull-down menu.
5. Click an object class such as Document (see below) to display more information.



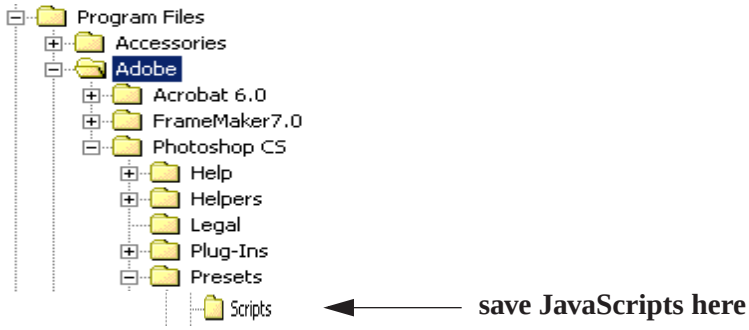
3.2.3 Viewing the JavaScript Environment

Because JavaScript is platform independent, there is no *native* script editor associated with it. Photoshop, however, provides a built-in run-time environment for executing JavaScripts. For more information on using JavaScript with Photoshop, see the JavaScript Reference Guide on the installation CD.

The singular advantage to writing JavaScripts is that the scripts run on any platform, regardless of the underlying hardware or operating system.

JavaScript is consequently the language of choice for developers not wishing to lock themselves into a single proprietary platform.

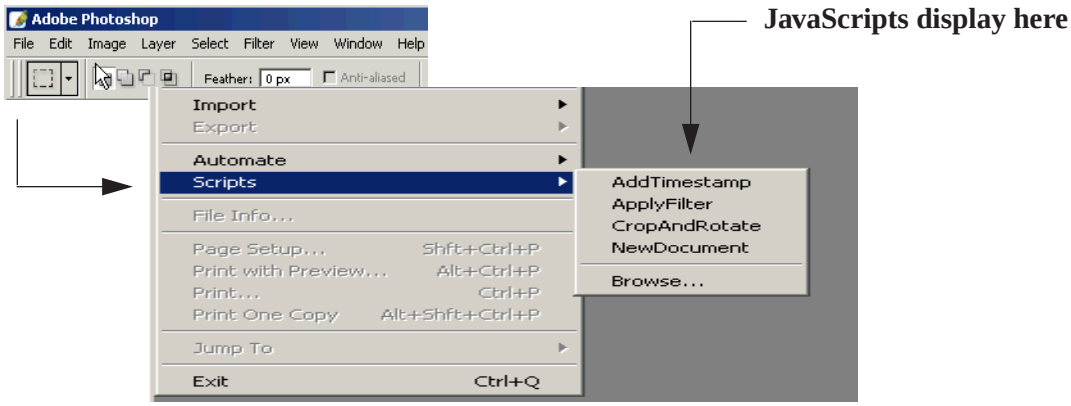
Use the text editor of your choice to create JavaScripts. To make scripts accessible to Photoshop, drag-and-drop your files directly into the "Presets>Scripts" folder.



Restart the Photoshop application to display JavaScripts in the Scripts menu.

The Scripts menu

The Scripts menu displays under the Photoshop File menu. When a Scripts item is selected, a dialog is presented from which you can select a JavaScript for execution.



A collection of JavaScripts comes pre-installed with Photoshop. These scripts display alongside the Scripts menu, as illustrated above.

You can use these scripts “out-of-the-box” or create your own. Use the “Browse” option to locate scripts you’ve created in other directories or to find scripts that reside on a network.

On both Mac and Windows, a JavaScript file must be saved as a text file with a '.js' file name extension.

Click a script to execute it. If there is an error encountered during script execution, an error dialog will be displayed containing the error message returned by the script.

If you hold down the option key (alt for Windows), a debug window displays.

NOTE: The "File>Scripts" menu displays JavaScripts only.

3.3 Your first Photoshop script

The traditional first project in any programming language is to display the message “Hello World!” In this section, we’ll create a new Photoshop document, then add a text item containing this message with examples in AppleScript, Visual Basic, VBScript and JavaScript.

3.3.1 AppleScript

1. Locate and open the Script Editor.
2. Below we’re going to revisit the "Hello, World!" AppleScript example from Chapter 2 with comments included. (We’ll expand on this sample code in the Advanced Scripting section that follows.)

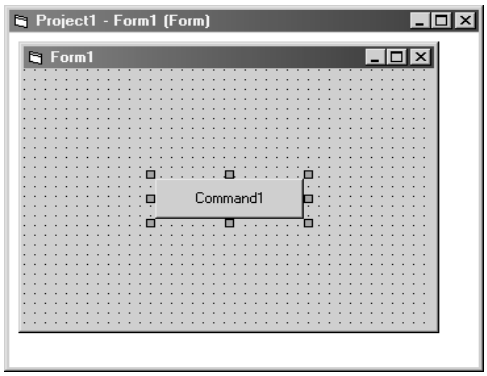
   For now, enter the following script. The lines preceded by “--” are comments. They’re included to document the operation of the script and it’s good style to include them in your own scripts. As you look through the script, you’ll see how to create, then address, each object. The AppleScript command `tell` indicates the object that will receive the next message we send.

```
-- Sample script to create a new text item and change its
-- contents.
tell application "Adobe Photoshop CS"

-- Create a new document and art layer.
     set docRef to make new document with properties ¬
        {width:3 as inches, height:2 as inches}
     set artLayerRef to make new art layer in docRef

-- Change the art layer to be a text layer.
     set kind of artLayerRef to text layer

-- Get a reference to the text object and set its contents.
     set contents of text object of artLayerRef to "Hello, World!"
end tell
```

3. Run the script. Photoshop will create a new document, add a new art layer, change the art layer’s type to text and set the text to “Hello, World!”

3.3.2 Visual Basic

1. Start Visual Basic and create a new project. Add the “Adobe Photoshop CS Object Library” reference to the project, as shown earlier. If you are using a built-in editor in a VBA application, skip to step 4.
2. Add a form to the project.
3. Create a new button on the form. Double-click the button to open the Code window.



4. Below we revisit the "Hello, World!" Visual Basic script from Chapter 2, with comments included. (We’ll expand on this example in the Advanced Scripting section that follows).

For now, enter the following code. The lines preceded by ' (single quotes) are comments, and will be ignored by the scripting system. They're included to describe the operation of the script. As you look through the script, you'll see how to create, then address each object.

```
Private Sub Command1_Click()

' Hello World Script
Dim appRef As New Photoshop.Application

' Remember current unit settings and then set units to
' the value expected by this script
Dim originalRulerUnits As Photoshop.PsUnits
originalRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = psInches

' Create a new 4x4 inch document and assign it to a variable.
Dim docRef As Photoshop.Document
Dim artLayerRef As Photoshop.ArtLayer
Dim textItemRef As Photoshop.TextItem
Set docRef = appRef.Documents.Add(4, 4)

' Create a new art layer containing text
Set artLayerRef = docRef.ArtLayers.Add
artLayerRef.Kind = psTextLayer

' Set the contents of the text layer.
Set textItemRef = artLayerRef.TextItem
textItemRef.Contents = "Hello, World!"

' Restore unit setting
appRef.Preferences.RulerUnits = originalRulerUnits

End Sub
```

5. Save the form.
6. Start Photoshop.
7. Return to Visual Basic and run the program. If you created a form, click the button you created earlier.
8. Run the script. Photoshop will create a new document, add a new art layer, change the art layer's type to text and set the text to "Hello, World!"

3.3.3 VBScript

You don't need to use Visual Basic to run scripts on Windows. Another way to script Photoshop is to use a VBA editor (such as the one that is included in Microsoft Word) or to use Windows Scripting Host.

Most Windows systems include Windows Scripting Host. If you do not have Windows Scripting Host or would like more information about Windows Scripting Host visit the Microsoft Windows Script Technologies Web site at http://msdn.microsoft.com/scripting/.

VBScript considerations

Both VBA and Windows Scripting Host use VBScript as their scripting language. The syntax for VBScript is very similar to the Visual Basic syntax. The three main differences relating to the scripts shown in this guide are:

– VBScript is not as strongly typed as Visual basic. In Visual Basic you say:

```
Dim aRef as Photoshop.ArtLayer
```

in VBScript you say:

```
Dim aRef
```

For VBScript simply omit the “as” and everything that comes after the "as" (in this case Photoshop.ArtLayer).

– VBScript does not support the “as New Photoshop.Application” form.

In Visual Basic you can retrieve the Application object as:

```
Dim appRef as New Photoshop.Application
```

In VBScript you write the following to retrieve the Application object:

```
Dim appRef
Set appRef = CreateObject("Photoshop.Application")
```

– VBScript does not support enumerations. Here's an example of how to set the extension type that can later be used to save a document.

```
Dim extType As Photoshop.PsExtensionType
extType = psUppercase
```

In Visual Basic, the values of the various enumerations are specified in a parenthesis after the enumeration name. For an enumeration value such as “psTextLayer (2)”, you would typically use the term “psTextLayer” (rather than "2") to refer to the kind of layer being described. For example:

```
artLayerRef.Kind = psTextLayer
```

VBScript, however, has no access to a type library; consequently, only the enumeration value "2" can be used -- not the term “psTextLayer”. For example:

```
artLayerRef.Kind = 2
```

Here's the "Hello, World!" script from Chapter 2, rewritten in VBScript with comments included.

```
' Hello World Script
Dim appRef
Set appRef = CreateObject( "Photoshop.Application" )

' Remember current unit settings and then set units to
' the value expected by this script
Dim originalRulerUnits
originalRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = 2

' Create a new 4x4 inch document and assign it to a variable.
Dim docRef
Dim artLayerRef
Dim textItemRef
Set docRef = appRef.Documents.Add(4, 4)

' Create a new art layer containing text
Set artLayerRef = docRef.ArtLayers.Add
artLayerRef.Kind = 2

' Set the contents of the text layer.
Set textItemRef = artLayerRef.TextItem
textItemRef.Contents = "Hello, World!"

' Restore unit setting
appRef.Preferences.RulerUnits = originalRulerUnits
```

To run this script create a text file and copy the script into it. Save the file with a "vbs" extension. Double-click the file to execute.

3.3.4 JavaScript

1. Locate and open the script editor of your choice.
2. Below we revisit the "Hello, World!" JavaScript example from Chapter 2 with comments included. (We'll expand on this code sample in the next section.)

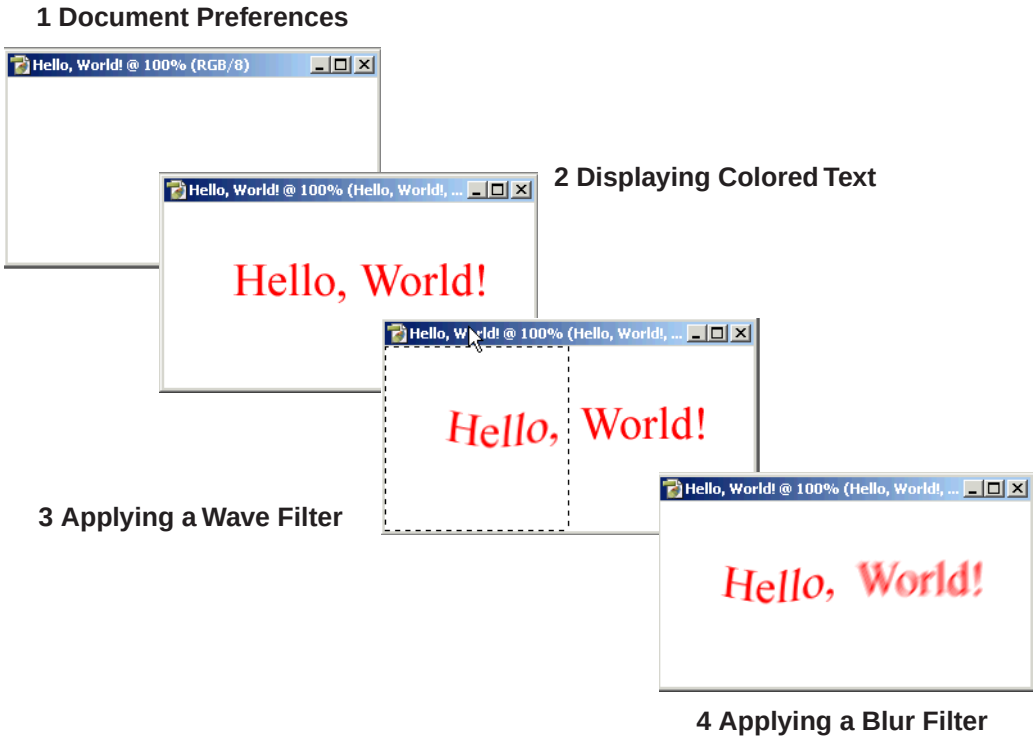
   For now, enter the following script. The lines preceded by "//" are comments. They're included to document the operation of the script and it's good style to include them in your own scripts. As you look through the script, you'll see how to create, then address, each object.

```
// Hello Word Script
// Remember current unit settings and then set units to
// the value expected by this script
var originalUnit = preferences.rulerUnits;
preferences.rulerUnits = Units.INCHES;

// Create a new 4x4 inch document and assign it to a variable
var docRef = app.documents.add( 4, 4 );

// Create a new art layer containing text
var artLayerRef = docRef.artLayers.add();
artLayerRef.kind = LayerKind.TEXT;

// Set the contents of the text layer.
var textItemRef = artLayerRef.textItem;
textItemRef.contents = "Hello, World!";

// Release references
docRef = null;
artLayerRef = null;
textItemRef = null;

// Restore original ruler unit setting
app.preferences.rulerUnits = originalUnit;
```

3.4 Advanced Scripting

Having familiarized yourself with the basic "Hello, World!" script, you are now ready to move on to more advanced scripting techniques such as configuring document preferences, applying color to text items, and rasterizing text so that *wrap* an d *blur* processing can be applied to words.

In this section, we describe sample code that takes you step-by-step through an extended script to successively produce the output illustrated below.



3.4.1 Advanced JavaScript

The following JavaScript code sample is broken down into four sections, the first of which deals with configuring document preferences.

NOTE: In JavaScript, the Photoshop Application object is already created for you behind the scenes. You do not have to make an explicit object reference to it.

Document Preferences

In this code segment, default configuration settings for the application are first saved into variables so that they can be restored later when the script completes. These are the default configurations you most probably set up using the File/Edit/Preferences dialog when you initially installed and configured Photoshop.

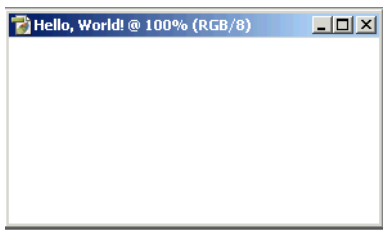
The script goes on to define new preferences for rulers and units and sets these to inches and pixels, respectively.

Dialog modes is set to "NO" so that the script runs without user intervention. Users will not, in other words, be required to press "OK" each time the script generates a new dialog for display.

Next, variables are declared that store document dimensions in inches and document resolution in pixels. Finally, a document object is created, if one does not already exist.

```
startRulerUnits = app.preferences.rulerUnits;
startTypeUnits = app.preferences.typeUnits;
startDisplayDialogs = app.displayDialogs;

app.preferences.rulerUnits = Units.INCHES;
app.preferences.typeUnits = TypeUnits.PIXELS;
app.displayDialogs = DialogModes.NO;

docWidthInInches = 4;
docHeightInInches = 2;
resolution = 72;

if (app.documents.length == 0)
    app.documents.add(docWidthInInches, docHeightInInches,
resolution);
```

This code produces the blank document displayed below.



Displaying Colored Text

Having generated a default document, you can now display colored text. To do so, first set a local reference to the current document. Then create a SolidColor object and assign RGB color values to it. After defining the text for the current layer, create an art layer of type TEXT.

NOTE: For a complete listing of all JavaScript properties, methods and constants (such as TEXT), please refer to the JavaScript Reference Guide, included as a separate document with Photoshop CS.

Next, set the text, position, size and color of the text layer. The content of the text layer is the expression "Hello, World!".

Notice that the position property of the text layer is an array whose values were chosen to roughly center the text in the dialog. A relatively large font size was chosen to increase the visibility of the text message. The color property is the SolidColor object created earlier, whose function is to display text in red.

```
docRef = app.activeDocument;

textColor = new SolidColor;
textColor.rgb.red = 255;
textColor.rgb.green = 0;
textColor.rgb.blue = 0;

helloWorldText = "Hello, World!";

newTextLayer = docRef.artLayers.add();

newTextLayer.kind = LayerKind.TEXT;

newTextLayer.textItem.contents = helloWorldText;
newTextLayer.textItem.position = Array(0.75, 1);
newTextLayer.textItem.size = 36;
newTextLayer.textItem.color = textColor;
```

This code snippet outputs "Hello, World!" in red.

Applying a Wave Filter

Now that text displays on your document, you're ready to apply some special effects. First, re-define the width and height of the document in pixels. Additionally, convert the text layer to pixels -- we do this because text is a vector graphic and we need a bitmap in order to manipulate the image.

Next create an array to specify the area to be selected for image manipulation. Notice that the array of points begins at the top left corner of the dialog and extends half way across the document. Other array values define vertical positioning.

With the width and height of the array thus defined, select the left side of the document. "Ants marching up the page" delimit the area selected.

You can now apply a wave filter to the selection. A truncated sin curve carries the text along for a roller-coaster-like ride.

```
docWidthInPixels = docWidthInInches * resolution;
docHeightInPixels = docHeightInInches * resolution;

newTextLayer.rasterize(RasterizeType.TEXTCONTENTS);

selRegion = Array(Array(0, 0),
                  Array(docWidthInPixels / 2, 0),
                  Array(docWidthInPixels / 2, docHeightInPixels),
                  Array(0, docHeightInPixels),
                  Array(0, 0));

docRef.selection.select(selRegion);

newTextLayer.applyWave(1, 1, 100, 5, 10, 100, 100,
                       WaveType.SINE, UndefinedAreas.WRAPAROUND, 0);
```

This code snippet manipulates and *bends* the text on the left side of the document.

Applying a MotionBlur Filter

Similar code can be used to blur the text in a document. Again create an array of points to designate an area of the document. This time the width is defined as the distance from the middle of the document to the far right side. The vertical positioning remains the same.

Select the right side of the document and apply a motion filter with parameters that define the angle and radius of the blur. Then remove the selection so that the "marching ants" disappear from the dialog.

To finish up, set application preferences back to their original values.

```
selRegion = Array(Array(docWidthInPixels / 2, 0),
                  Array(docWidthInPixels, 0),
                  Array(docWidthInPixels, docHeightInPixels),
                  Array(docWidthInPixels / 2, docHeightInPixels),
                  Array(docWidthInPixels / 2, 0));

docRef.selection.select(selRegion);

newTextLayer.applyMotionBlur(45, 5);

docRef.selection.deselect();

app.preferences.rulerUnits = startRulerUnits;
app.preferences.typeUnits = startTypeUnits;
app.displayDialogs = startDisplayDialogs;
```

This code snippet removes the "marching ants" and *blurs* the text on the right side of the document.

3.4.2 Advanced Visual Basic

The following Visual Basic code sample is broken down into four sections, the first of which deals with configuring document preferences.

Document Preferences

In this code segment, variables are declared and a Photoshop Application object is created.

Default configuration settings for the application are saved into variables so that they can be restored later when the script completes. These are the default configurations you most probably set up using the File/Edit/Preferences dialog when you initially installed and configured Photoshop.

The script goes on to define new preferences for rulers and units and sets these to inches and pixels, respectively. The psDisplayNoDialogs enumeration is specified so that the script runs without user intervention. Users will not, in other words, be required to press "OK" each time the script generates a new dialog for display.

Next, variables are declared that store document dimensions in inches and document resolution in pixels. A display resolution is declared and the text "Hello, World!" is assigned to a string variable.

Finally, a document object is created, if one does not already exist.

```
 Dim startRulerUnits As Photoshop.PsUnits
Dim startTypeUnits As Photoshop.PsTypeUnits
Dim startDisplayDialogs As Photoshop.PsDialogModes
Dim docWidthInInches As Integer
Dim docHeightInInches As Integer
 Dim docWidthInPixels As Integer
Dim docHeightInPixels As Integer
Dim resolution As Integer
Dim helloWorldStr As String
Dim app As Photoshop.Application
Dim docRef As Photoshop.Document
Dim textColor As Photoshop.SolidColor
Dim newTextLayer As Photoshop.ArtLayer
Dim Preferences As Photoshop.Preferences

Set app = New Photoshop.Application

startRulerUnits = app.Preferences.RulerUnits
startTypeUnits = app.Preferences.TypeUnits
startDisplayDialogs = app.DisplayDialogs
```

```
app.Preferences.RulerUnits = Photoshop.PsUnits.psInches
app.Preferences.TypeUnits =_
  Photoshop.PsTypeUnits.psTypePixels
app.DisplayDialogs =_
  Photoshop.PsDialogModes.psDisplayNoDialogs

docWidthInInches = 4
docHeightInInches = 2
resolution = 72
helloWorldStr = "Hello, World!"

If app.Documents.Count = 0 Then
    app.Documents.Add docWidthInInches, docHeightInInches,_
     resolution, helloWorldStr
End If
```

This code produces the blank document displayed below.

Displaying Colored Text

Having generated a default document, you can now display colored text. To do so, first set a local reference to the current document. Then create a SolidColor object and assign RGB color values to it. You are now ready to create an art layer of type `psTextLayer`.

NOTE: For a complete listing of all Visual Basic properties, methods and constants (such as `psTextLayer`), please refer to the Visual Basic Reference Guide, included as a separate document with Photoshop CS.

Next, set the text, position, size and color of the text layer. The content of the text layer is the string variable `helloWorldStr` declared in the previous section.

Notice that the position property of the text layer is an array whose values were chosen to roughly center the text in the dialog. A relatively large font size was chosen to increase the visibility of the text message. The color property is the SolidColor object created earlier, whose function is to display text in red.

```
Set docRef = app.ActiveDocument

Set textColor = New Photoshop.SolidColor
textColor.RGB.Red = 255
textColor.RGB.Green = 0
textColor.RGB.Blue = 0

Set newTextLayer = docRef.ArtLayers.Add()
newTextLayer.Kind = Photoshop.PsLayerKind.psTextLayer

newTextLayer.TextItem.Contents = helloWorldStr
newTextLayer.TextItem.Position = Array(0.75, 1)
newTextLayer.TextItem.Size = 36
newTextLayer.TextItem.Color = textColor
```

This code snippet outputs "Hello, World!" in red.

Applying a Wave Filter

Now that text displays on your document, you're ready to apply some special effects. First, re-define the width and height of the document in pixels. Additionally, convert the text layer to pixels -- we do this because text is a vector graphic and we need a bitmap in order to manipulate the image.

Next create an array to specify the area to be selected for image manipulation. Notice that the array of points begins at the top left corner of the dialog and extends half way across the document. Other array values define vertical positioning.

With the width and height of the array thus defined, select the left side of the document. "Ants marching up the page" delimit the area selected.

You can now apply a wave filter to the selection. A truncated sin curve carries the text along for a roller-coaster-like ride.

```
docWidthInPixels = docWidthInInches * resolution
docHeightInPixels = docHeightInInches * resolution

newTextLayer.Rasterize
  Photoshop.PsRasterizeType.psTextContents

docRef.Selection.Select Array(Array(0, 0), _
  Array(docWidthInPixels / 2, 0), _
  Array(docWidthInPixels / 2, docHeightInPixels), _
  Array(0, docHeightInPixels), Array(0, 0))

newTextLayer.ApplyWave 1, 1, 100, 5, 10, 100, 100, _
  Photoshop.PsWaveType.psSine, _
     Photoshop.PsUndefinedAreas.psWrapAround, 0
```

This code snippet manipulates and *bends* the text on the left side of the document.

Applying a MotionBlur Filter

Similar code can be used to blur the text in a document. Again create an array of points to designate an area of the document. This time the width is defined as the distance from the middle of the document to the far right side. The vertical positioning remains the same.

Select the right side of the document and apply a motion filter with parameters that define the angle and radius of the blur. Then remove the selection so that the "marching ants" disappear from the dialog.

To finish up, set application preferences back to their original values.

```
docRef.Selection.Select Array(Array(docWidthInPixels / 2, 0), _
      Array(docWidthInPixels, 0), _
      Array(docWidthInPixels, docHeightInPixels), _
      Array(docWidthInPixels / 2, docHeightInPixels), _
      Array(docWidthInPixels / 2, 0))

    newTextLayer.ApplyMotionBlur 45, 5

    docRef.Selection.Deselect

    app.Preferences.RulerUnits = startRulerUnits
    app.Preferences.TypeUnits = startTypeUnits
    app.DisplayDialogs = startDisplayDialogs
```

This code snippet removes the "marching ants" and *blurs* the text on the right side of the document.

3.4.3 Advanced AppleScript

The following AppleScript code sample is broken down into four sections, the first of which deals with configuring document preferences.

Document Preferences

In this code segment, a Photoshop Application object is activated. Default configuration settings for the application are saved into variables so that they can be restored later when the script completes. These are the default configurations you most probably set up using the File/Edit/Preferences dialog when you initially installed and configured Photoshop.

The script goes on to define new preferences for rulers and units and sets these to inches and pixels, respectively.

Dialog modes is set to "never" so that the script runs without user intervention. Users will not, in other words, be required to press "OK" each time the script generates a new dialog for display.

Next, variables are declared that store document dimensions in inches and document resolution in pixels. A display resolution is declared and the text "Hello, World!" is assinged to a string variable.

Finally, a document object is created, if one does not already exist.

```
tell application "Adobe Photoshop CS"

    activate

    set theStartRulerUnits to ruler units of settings
    set theStartTypeUnits to type units of settings
    set theStartDisplayDialogs to display dialogs

    set ruler units of settings to inch units
    set type units of settings to pixel units
    set display dialogs to never

    set theDocWidthInInches to 4
    set theDocHeightInInches to 2
    set theDocResolution to 72
    set theDocString to "Hello, World!"

    if (count of documents) is 0 then
       make new document with properties ¬
       {width:theDocWidthInInches, height:theDocHeightInInches,¬
          resolution:theDocResolution, name:theDocString}
    end if
```

This code produces the blank document displayed below.

Displaying Colored Text

Having generated a default document, you can now display colored text. To do so, first set a local reference to the current document. Then create a SolidColor object and assign RGB color values to it. After defining the text for the current layer, create an art layer of type `text layer`.

NOTE: For a complete listing of all AppleScript properties, methods and constants (such as `text layer`), please refer to the AppleScript Reference Guide, included as a separate document with Photoshop CS.

Next, set the text, position, size and color of the text layer. The contents of the text layer is the expression "Hello, World!".

Notice that the position property of the text layer is an array whose values were chosen to roughly center the text in the dialog. A relatively large font size was chosen to increase the visibility of the text message. The color property is `theTextColor` created earlier, whose function is to display text in red.

```
set theDocRef to the current document

set theTextColor to {class:RGB color, red:255, green:0, blue:0}

set theTextLayer to make new art layer in theDocRef with¬
   properties {kind:text layer}

set contents of text object of theTextLayer to "Hello, World!"
set size of text object of theTextLayer to 36
set position of text object of theTextLayer to {0.75, 1}
set stroke color of text object of theTextLayer to theTextColor
```

This code snippet outputs "Hello, World!" in red.

Applying a Wave Filter

Now that text displays on your document, you're ready to apply some special effects. First, re-define the width and height of the document in pixels. Additionally, convert the text layer to pixels -- we do this because text is a vector graphic and we need a bitmap in order to manipulate the image.

Next create an array to specify the area to be selected for image manipulation. Notice that the array of points begins at the top left corner of the dialog and extends half way across the document. Other array values define vertical positioning.

With the width and height of the array thus defined, select the left side of the document. "Ants marching up the page" delimit the area selected.

You can now apply a wave filter to the selection. A truncated sin curve carries the text along for a roller-coaster-like ride.

```
set theDocWidthInPixels to theDocWidthInInches *¬
   theDocResolution
set theDocHeightInPixels to theDocHeightInInches *¬
   theDocResolution

rasterize theTextLayer affecting text contents

set theSelRegion to {{0, 0}, ¬
   {theDocWidthInPixels / 2, 0}, ¬
   {theDocWidthInPixels / 2, theDocHeightInPixels}, ¬
   {0, theDocHeightInPixels}, ¬
   {0, 0}}

select theDocRef region theSelRegion combination type replaced

filter current layer of theDocRef using wave filter ¬
      with options {class:wave filter, number of generators:1 ¬
      , minimum wavelength:1, maximum wavelength:100, ¬
          minimum amplitude:5, maximum amplitude:10 ¬
      , horizontal scale:100, vertical scale:100 ¬
      , wave type:sine, undefined areas:repeat edge pixels,¬
         random seed:0}
```

This code snippet manipulates and *bends* the text on the left side of the document.

Applying a MotionBlur Filter

Similar code can be used to blur the text in a document. Again create an array of points to designate an area of the document. This time the width is defined as the distance from the middle of the document to the far right side. The vertical positioning remains the same.

Select the right side of the document and apply a motion filter with parameters that define the angle and radius of the blur. Then remove the selection so that the "marching ants" disappear from the dialog.

To finish up, set application preferences back to their original values.

```
    set theSelRegion to {{theDocWidthInPixels / 2, 0},
       {theDocWidthInPixels, 0}, ¬
       {theDocWidthInPixels, theDocHeightInPixels}, ¬
       {theDocWidthInPixels / 2, theDocHeightInPixels}, ¬
       {theDocWidthInPixels / 2, 0}}

    select theDocRef region theSelRegion combination type replaced

    filter current layer of theDocRef using motion blur ¬
       with options {class:motion blur, angle:45, radius:5}

    deselect theDocRef

    set ruler units of settings to theStartRulerUnits
    set type units of settings to theStartTypeUnits
    set display dialogs to theStartDisplayDialogs

end tell
```

This code snippet removes the "marching ants" and *blurs* the text on the right side of the document.

3.5 Object references

The remainder of this document elaborates on the scripting concepts and Object Model components described in the previous code samples.

The objects in your scripts are arranged in a hierarchy that mirrors the Object Model. When you send a command to a Photoshop object, you need to make sure you send the message to the right object. The way you identify an object varies with the scripting language used.

3.5.1 AppleScript

AppleScript uses object references to identify the target object for commands. When working with Photoshop you can identify each item in an object reference using either index or name form. For example, if you have a single document, named “My Document”, open, you could target the document’s first layer, named “Cloud Layer” with either line:

```
layer 1 of document 1
```

or

```
layer "cloud layer" of document "My Document"
```

NOTE: When scripting Phothshop a document's index is not always the same as its stacking order in the user interface. It is possible for `document 1` to not be the front-most document. For this reason Photoshop will always return object references identifying documents by name. It is recommended that you always use the name form when identifying documents in your scripts.

An object's index or name also may change as a result of manipulating other objects. For example, when a new art layer is created in the document, it will become the first layer, and the layer that was previously the first layer is now the 2nd layer. Therefore, any references made to layer 1 of current document will now refer to the new layer.

Consider the following sample script:

```
1. tell application "Adobe Photoshop CS"
2.    activate
```

```
3.   set newDocument to make new document with properties ¬
        { width: inches 2, height: inches 3}
4.   set layerRef to layer 1 of current document
5.   make new art layer in current document
6.   set name of layerRef to "My layer"
7. end tell
```

This script will not set the name of the layer referenced on the fourth line of the script. Instead it will set the name created on line five. Try referencing the objects by name as shown below:

```
1. tell application "Adobe Photoshop CS"
2.   activate
3.   set newDocument to make new document with properties ¬
        { width: inches 2, height: inches 3}
4.   make new art layer in current document with properties {name: "L1" }
5.   make new art layer in current document with properties {name: "L2" }
6.   set name of art layer "L1" of current document to "New Layer 1"
7. end tell
```

3.5.2 Visual Basic and JavaScript

Object references in Visual Basic and JavaScript are fixed and remain valid until disposed or until the host object goes away.

The following example shows how to create 2 layers and then rename the first one in Visual Basic.

```
Dim appRef As Photoshop.Application
Dim docRef As Photoshop.Document
Dim layer1Ref As Photoshop.ArtLayer
Dim layer2Ref As Photoshop.ArtLayer

' Set ruler units and create a new document.
Set appRef = New Photoshop.Application
originalRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = psInches
Set docRef = appRef.Documents.Add(4, 4, 72, "My New Document")

' Create 2 new layers and store their return references.
Set layer1Ref = docRef.ArtLayers.Add()
Set layer2Ref = docRef.ArtLayers.Add()

' Change the name of the first layer that was created.
layer1Ref.Name = "This layer was first"

'restore unit values
appRef.Preferences.RulerUnits = originalRulerUnits
```

The following example shows how to create 2 layers and then rename the first one in JavaScript.

```
// set ruler units and create new document
originalRulerUnits = app.preferences.rulerUnits
app.preferences.rulerUnits = Units.INCHES;
app.documents.add(4,4,72,"My New Document");
docRef = app.activeDocument;
layer1Ref = docRef.artLayers.add();
layer2Ref = docRef.artLayers.add();
layer1Ref.name = "This layer was first";

// restore unit setting
app.preferences.rulerUnits = originalRulerUnits;
```

3.6 Working with units

Photoshop provides two rulers for use when working on a document — a graphics ruler used for most graphical layout measurements and a type ruler which is active when using the type tool. The unit types for these two rulers are set using the `ruler units (RulerUnits/rulerUnits)` and `type units (TypeUnits/typeUnits)`, respectively. These settings correspond to those found in the Photoshop preference dialog under "Edit >Preferences > Units & Rulers."

The graphics ruler is used for most operations on a document where height, width, or position are specified. The type ruler is used when operating on text items, such as when setting leading or indent values. By changing the settings for each ruler you can work with documents in the measurement system that make the most sense for the project at hand.

3.6.1 Unit values

Photoshop uses unit values for certain properties and parameters. Scripting comments concerning Photoshop objects and properties note where unit values are used.

Because of scripting language differences, the way you provide a unit value in a script depends on the language you are using. All languages support plain numbers for unit values. These values are treated as being of the type currently specified for the appropriate ruler.

For example, if the ruler units are currently set to inches and the following Visual Basic statement is executed:

```
docRef.ResizeImage 3,3
```

the document's image is resized to 3 inches by 3 inches. If the ruler units were set to pixels, the image would be 3 pixels by 3 pixels, which is probably not what was intended. To ensure that your scripts produce the expected results you should check and set the ruler units to the type

appropriate for your script. After executing a script the original values of the rule settings should be restored if changed in the script. See section 3.6.3, “Changing ruler and type units” on page 61 for directions on setting unit values.

AppleScript unit considerations

AppleScript provides an additional way of working with unit values. You can provide values with an explicit unit type where unit values are used. When a typed value is provided its type overrides the ruler’s current setting.

For example, to create a document which is 4 inches wide by 5 inches high you would write:

```
make new document with properties {width:inches 4, ¬
    height:inches 5}
```

The values returned for a Photoshop property which used units will be returned as a value of the current ruler type. Getting the height of the document created above:

```
set docHeight to height of current document
```

would return a value of 5.0, which represents 5 inches based on the current ruler settings.

In AppleScript, you can optionally ask for a property value as a particular type.

```
set docHeight to height of current document as points
```

This would return a value of 360 (5 inches x 72 points per inch).

IMPORTANT: *Because Photoshop is a pixel-oriented application you may not always get back the same value as you pass in when setting a value. For example, if Ruler Units is set to mm units, and you create a document that is 30 x 30, the value returned for the height or width will be 30.056 if your document resolution is set to 72 ppi. The scripting interface assumes settings are measured by ppi.*

The length unit value types available AppleScript use are listed below:

TABLE 3.1 AppleScript Length Unit Values

inches	millimeters
feet	centimeters
yards	meters
miles	kilometers
points	picas
traditional points	traditional picas
ciceros	

The `points` and `picas` unit value types are PostScript points, with 72 points per inch. The `traditional points` and `traditional picas` unit value types are based on classical type setting values, with 72.27 points per inch.

When working with unit values, it is possible to convert, or coerce, a unit value from one value type to another. For example, the following script will convert a point value to an inch value.

```
set pointValue to points 72
set inchValue to pointValue as inches
```

When this script is run the variable `inchValue` will contain inches 1 , which is 72 points converted to inches. This conversion ability is built in to the AppleScript language.

To use a unit value in a calculation it is necessary to first convert the value to a number (unit value cannot be used directly in calculations). To multiply an inch value write:

```
set newValue to (inchValue as number) * someValue
```

Special unit value types

The unit values used by Photoshop are length units, representing values of linear measurement. Support is also included for pixel and percent unit values. These two unit value types are not, strictly speaking, length values but are included because they are used extensively by Photoshop for many operations and values.

NOTE: In AppleScript you can get and set values as pixels or percent as you would any other unit value type. You cannot, however, convert a pixel or percent value to another length unit value as you can with other length value types. Trying to run the following script will result in an error.

```
set pixelValue to pixels 72
-- Next line will result in a coercion error when run
set inchValue to pixelValue as inches
```

3.6.2 Unit value useage

The following two tables list the properties of the classes and parameters of commands that are defined to use unit values. Unit values for these properties and parameter, with the exception of some text item properties, are based the graphics ruler setting.

TABLE 3.2 Object Properties

Object	AppleScript Properties	Visual Basic Properties	JavaScript Properties
Document	height width	Height Width	height width
EPS open options	height width	Height Width	height width
PDF open options	height width	Height Width	height width
lens flare open options	height width	Height Width	height width
offset filter	horizontal offset vertical offset	HorizontalOffset VerticalOffset	horizontalOffset verticalOffset
Text Item	baseline shift* first line indent* height hyphenation zone* leading* left indent* position right indent* space before* space after* width	BaselineShift* FirstLineIndent* Height HyphenationZone* Leading* LeftIndent* Position RightIndent* SpaceBefore* SpaceAfter* Width	baselineShift* firstLineIndent* height hyphenationZone* leading* leftIndent* position rightIndent* spaceBefore* spaceAfter* width

* Unit values based on type ruler setting

TABLE 3.3 Command Parameters

AppleScript	Visual Basic	JavaScript
crop (bounds, height, width)	Document.Crop (Bounds, Height, Width)	document.crop (bounds, height, width)
resize canvas (height, width)	Document.ResizeCanvas (Height, Width)	document.resizeCanvas (height, width)
resize image (height, width)	Document.ResizeImage (Height, Width)	document.resizeImage (height, width)
contract (by)	Selection.Contract (By)	selection.contract (by)
expand (by)	Selection.Expand (By)	selection.expand (by)
feather (by)	Selection.Feather (By)	selection.feather (by)
select border (width)	Selection.SelectBorder (Width)	selection.selectBorder (width)
translate (delta x, delta y)	Selection.Translate (DeltaX, DeltaY)	selection.translate (deltaX, deltaY)
translate boundary (delta x, delta y)	Selection.TranslateBoundary (DeltaX, DeltaY)	selection.translateBoundary (deltaX, deltaY)

3.6.3 Changing ruler and type units

The unit type settings of the two Photoshop rulers control how numbers are interpreted when dealing with properties and parameters that support unit values. Be sure to set the ruler units as needed at the beginning of your scripts and save and restore the original ruler settings when your script has completed.

In AppleScript `ruler units` and `type units` are properties of the `settings-object`, accessed through the Application object's `settings` property as shown below.

```
set ruler units of settings to inch units
set type units of settings to pixel units
set point size of settings to postscript size
```

In Visual Basic and JavaScript `ruler units` and `type units` are properties of the `Preferences`, accessed through the application object's preferences property as shown below.

VB:

```
appRef.Preferences.RulerUnits = PsInches
appRef.Preferences.TypeUnits = PsTypePixels
appRef.Preferences.PointSize = PsPostScriptPoints
```

JS:

```
app.preferences.rulerUnits = Units.INCHES;
app.preferences.typeUnits = TypeUnits.PIXELS;
app.preferences.pointSize = PointType.POSTSCRIPT;
```

IMPORTANT: *Remember to reset the unit settings back to the original values at the end of a script.*

3.7 Executing JavaScripts from AS or VB

You may want to take advantage of the platform-independence of JavaScript by running scripts from AppleScript or Visual Basic.

For AppleScript, use `do javascript`.

For Visual Basic, use either the Application's `DoJavaScript` or `DoJavaScriptFile` method. `DoJavaScript` takes a string, which is the JavaScript code to execute. `DoJavaScriptFile` opens a file that contains the JavaScript code as illustrated below:

AS:

```
set scriptFile to "myscript" as alias
do javascript scriptFile
```

VB:

```
Dim appRef As Photoshop.Application
Set appRef = CreateObject("Photoshop.Application")
appRef.DoJavaScriptFile ("D:\\Scripts\\MosaicTiles.js")
```

3.7.1 Passing arguments to JavaScript

You can also pass arguments to JavaScript from either AppleScript or Visual Basic by using the `with arguments (Arguments)` parameter. The parameter takes an array to pass any values.

For example, save the following JavaScript to a file on your machine:

```
alert( "You passed " + arguments.length + " arguments" );
for ( i = 0; i < arguments.length; ++i )
{
    alert( arguments[i].toString() )
}
```

To pass arguments from AppleScript try this:

```
tell application "Adobe Photoshop CS"
    make new document
    do javascript (alias <a path to the JavaScript shown above>) ¬
       with arguments {1, "test text", (file <a path to a file>),¬
       current document}
end tell
```

To do the same thing in VB, write:

```
Dim appRef As Photoshop.Application
Set appRef = CreateObject("Photoshop.Application")
appRef.DoJavaScriptFile "C:\scripts-temp\test.js", _
    Array(1, "text text", appRef.ActiveDocument)
```

When running JavaScript from AppleScript or Visual Basic you can also control the debugging state. To do this, use the `show debugger (ExecutionMode)` argument. The values of this argument include:

- `never (NeverShowDebugger)`: This option will disable debugging from the JavaScript. Any error that occurs in the JavaScript will result in a JavaScript exception being thrown. Note that you can catch JavaScript exceptions in your script; see the JavaScript Reference Guide for more information on how to handle JavaScript exceptions. When you use this option the JavaScript command “debugger();” will be ignored.
- `on runtime error (DebuggerOnError)`: This option will automatically stop the execution of your JavaScript when a runtime error occurs and show the JavaScript debugger. When you use this option the JavaScript command “debugger();” will stop the JavaScript and display the JavaScript debugger.
- `before running (BeforeRunning)`: This option will show the JavaScript debugger at the beginning of your JavaScript. When you use this option the JavaScript command “debugger();” will stop the JavaScript and display the JavaScript debugger.

3.7.2 Executing one-line JavaScripts

You can also execute simple JavaScripts directly without passing a file as shown in the following examples.

AS:

```
do javascript "alert('alert text');"
```

VB:

```
objApp.DoJavaScript ("alert('alert text');")
```

3.8 The Application object

AppleScript and Visual Basic scripts can target multiple applications so the first thing you should do in your script is target Photoshop.

By using the properties and commands of the Application object, you can work with global Photoshop settings, open documents, execute actions, and exercise other Photoshop functionality.

Targeting the Application object

To target the Photoshop application in AppleScript, you must use a `tell..end tell` block. By enclosing your Photoshop commands in the following statement, AppleScript will understand you are targeting Photoshop.

```
tell application "Adobe Photoshop CS"
```

…

```
end tell
```

In Visual Basic , you create and use a reference to the Application. Typically, you would write:

```
Set appRef = CreateObject("Photoshop.Application")
```

In JavaScript, there is no need for an application object and therefore, all properties and methods of the application are accessible without any qualification. To get the active Photoshop document in JavaScript, write:

```
var docRef = app.activeDocument;
```

Once you have targeted your application, you are ready to work with the properties and commands of the application object.

The active document

Because “document 1” does not always indicate the front-most document, it’s recommended that your scripts set the current or active document before executing any other commands. To do this, use the “current document (`ActiveDocument/activeDocument`)” property on the application object.

AS: `set docRef to current document`

VB: `Set docRef = appRef.ActiveDocument`

JS: `docRef = app.activeDocument;`

You can also switch back and forth between documents by setting the active document.

AS: `set current document to document "My Document"`

VB: `appRef.ActiveDocument = appRef.Documents("My Document")`

JS: `app.activeDocument = app.documents["My Document"];`

Application preferences

The application object contains a property for Photoshop preferences. The preferences property is itself an object and has many properties. The name of the preferences object for the three languages is the following:

AS: `settings`

VB: `Preferences`

JS: `preferences`

The properties in the preferences object correlate to the preferences found by displaying the Photoshop "Preferences" dialog in the user interface (select the "Edit > Preferences" menu in Photoshop).

Display dialogs

It is important to be able to control dialogs properly from a script. If a dialog is shown your script stops until a user dismisses the dialog. This is normally fine in an interactive script that expects a user to be sitting at the machine. But if you have a script that runs in an unsupervised (batch) mode you do not want dialogs to be displayed and stop your script.

Using the `display dialogs` (`DisplayDialogs/displayDialogs`) property on the application object you can control whether or not dialogs are displayed.

If you set `display dialogs` to `always ( psDisplayAllDialogs/ALL )`, Photoshop will show all user related dialogs. This is typically not what you want.

If you set `display dialogs` to `error dialogs (DisplayErrorDialogs/ERROR)`, then only dialogs related to errors are shown. You would typically use this setting when you are developing a script or if your script is an interactive one that expects a user to be sitting at the machine while running the script.

If you set `display dialogs` to `never ( DisplayNoDialogs/NO ),` then no dialogs are shown. If an error occurs it will be returned as an error to the script. See section 2.10.4, "Error handling" on page 29 for more information on catching errors.

Opening a document

When using the open command there are a number of specifiable options. These options are grouped by file type in the provided open options classes. Because the type and contents of the file you are working on affects how it is opened, some of the option values may not always be applicable. It also means that many of the option values do not have well defined default values.

The best way to determine what values can or should be used for open is to perform an open command from the user interface. You can then copy the value from the options dialog to your script. You should perform a complete open operation because there can be multiple dialogs presented before the document is actually opened. If you cancel one of the open dialogs without completing the operation you could miss seeing a dialog which contains values needed in your script.

Specifying file formats to open

Because Photoshop supports many different file formats, the `Open` command lets you specify the format of the document you are opening. If you do not specify the format, Photoshop will infer the type of file for you. Here's how to open a document using its default type:

AS:

```
set theFile to alias "MyFile.psd"
open theFile
```

VB:

```
fileName = "C:\MyFile.psd"
Set docRef = appRef.Open(fileName)
```

JS:

```
var fileRef = new File("//MyFile.psd");
var docRef = app.open (fileRef);
```

Notice that in JavaScript, you must create a `File` object, and it gets passed into the open command. See the JavaScript file documentation for more information.

Some formats require extra information when opening. When you open a Generic EPS, Generic PDF, Photo CD or Raw image you have to provide additional information to the open command.

Do this by using the various `open options` classes:

- `EPS Open Options (EPSOpenOptions/EPSOpenOptions)`
- `PDF Open Options (PDFOpenOptions/PDFOpenOptions)`
- `Photo CD Open Options (PhotoCDOpenOptions/PhotoCDOpenOptions)`
- `raw format Options (RawFormatOpenOptions/RawFormatOpenOptions)`

The following example shows how to open a generic PDF document.

AS:

```
tell application "Adobe Photoshop CS"
    set myFilePath to alias < a file path >
    open myFilePath as PDF with options ¬
       {class:PDF open options, height:pixels 100, ¬
          width:pixels 200, mode:RGB, resolution:72, ¬
             use antialias:true, page:1, ¬
                constrain proportions:false}
end tell
```

VB:

```
Dim appRef As Photoshop.Application
Set appRef = CreateObject("Photoshop.Application")

'Remember unit settings; and set to values expected by this script
Dim originalRulerUnits As Photoshop.PsUnits
originalRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = psPixels

'Create a PDF option object
Dim pdfOpenOptionsRef As Photoshop.PDFOpenOptions
Set pdfOpenOptionsRef = CreateObject("Photoshop.PDFOpenOptions")
pdfOpenOptionsRef.AntiAlias = True
pdfOpenOptionsRef.Height = 100
pdfOpenOptionsRef.Width = 200
pdfOpenOptionsRef.mode = psOpenRGB
pdfOpenOptionsRef.Resolution = 72
pdfOpenOptionsRef.ConstrainProportions = False

'Now open the file
Dim docRef As Photoshop.Document
Set docRef = appRef.Open(< a file path>, pdfOpenOptionsRef)

'Restore unit setting
appRef.Preferences.RulerUnits = originalRulerUnits
```

JS:

```
// Set the ruler units to pixels
var originalRulerUnits = app.preferences.rulerUnits;
app.preferences.rulerUnits = Units.PIXELS;
```

```
// Get a reference to the file that we want to open
var fileRef = new File(  < a file path > );

// Create a PDF option object
var pdfOpenOptions = new PDFOpenOptions;
pdfOpenOptions.antiAlias = true;
pdfOpenOptions.height = 100;
pdfOpenOptions.width = 200;
pdfOpenOptions.mode = OpenDocumentMode.RGB;
pdfOpenOptions.resolution = 72;
pdfOpenOptions.constrainProportions = false;

// Now open the file
app.open( fileRef, pdfOpenOptions );

// restore unit settings
app.preferences.rulerUnits = originalRulerUnits;
```

Because Photoshop cannot save all of the format types that it can open, the open document types may be different from the save document types.

3.9 Document object

After you target the Photoshop application, the next object you will likely target is the `Document` object. The `Document` object can represent any open document in Photoshop.

For example, you could use the `Document` object to get the active layer, save the current document, then copy and paste within the active document or between different documents.

3.9.1 Saving documents and save options

Photoshop lets you work with various file formats. It is important to note, however, that the `Open` and `Save` formats are not identical.

Also note that some formats available in scripting require you to install optional file formats. The optional formats are:

- Alias PIX
- Electric Image
- SGI RGB
- Wavefront RLA
- SoftImage

When using the save command there are a number of specifiable options. These options are grouped by file type in the provided save options classes. Because the type and contents of the file you are working on affects how it is saved, some of the option values may not always be applicable. It also means that many of the option values do not have well defined default values.

The best way to determine what values can or should be used for save is to perform a save command from the user interface and then copy the value from the options dialog to your script. You should perform a complete save operation because there can be multiple dialogs presented before the document is saved. If you cancel one of the save dialogs without completing the operation you could miss a dialog containing values needed in your script.

There are many objects that allow you to specify how you want to save your document. For example, to save a file as a JPEG file, you would use the `JPEG save options` (`JPEGSaveOptions/JPEGSaveOptions`) class as shown below.

AS:

```
tell application "Adobe Photoshop CS"
     make new document
     set myOptions to {class:JPEG save options, ¬
        embed color profile:false, format options: standard, ¬
           matte: background color matte,}
     save current document in file myFile as JPEG with options ¬
        myOptions appending no extension without copying
end tell
```

VB:

```
Dim appRef As New Photoshop.Application
Set jpgSaveOptions = CreateObject("Photoshop.JPEGSaveOptions")
jpgSaveOptions.EmbedColorProfile = True
jpgSaveOptions.FormatOptions = psStandardBaseline
jpgSaveOptions.Matte = psNoMatte
jpgSaveOptions.Quality = 1
appRef.ActiveDocument.SaveAs "c:\temp\myFile2", _
     Options:=jpgSaveOptions, _
        asCopy:=True, extensionType:=psLowercase
```

JS:

```
jpgFile = new File( "/Temp001.jpeg" );
jpgSaveOptions = new JPEGSaveOptions();
jpgSaveOptions.embedColorProfile = true;
jpgSaveOptions.formatOptions = FormatOptions.STANDARDBASELINE;
jpgSaveOptions.matte = MatteType.NONE;
jpgSaveOptions.quality = 1;
app.activeDocument.saveAs(jpgFile, jpgSaveOptions, true,
    Extension.LOWERCASE);
```

3.9.2 Document information

A Photoshop document can be associated with additional information such as the author via the “File > File Info” menu.

The information found in this menu-item is handled by the `info (DocumentInfo)` object.To change document information, reference the `info` object and set its properties as shown below.

AS:

```
set docInfoRef to info of current document
set copyrighted of docInfoRef to copyrighted work
set owner url of docInfoRef to "http://www.adobe.com"
```

VB:

```
Set docInfoRef = docRef.Info
docInfoRef.Copyrighted = psCopyrightedWork
docInfoRef.OwnerUrl = "http://www.adobe.com"
```

JS:

```
docInfoRef = docRef.info;
docInfoRef.copyrighted = CopyrightedType.COPYRIGHTEDWORK;
docInfoRef.ownerUrl = "http://www.adobe.com";
```

3.9.3 Document manipulation

The `Document` object is used to make modifications to the document image. By using the `Document` object you can crop, rotate or flip the canvas, resize the image or canvas, and trim the Image.

Because unit values are passed in when resizing an image, it is recommended that you first set your ruler units prior to resizing. See section 3.6.3, “Changing ruler and type units” on page 61 for more information.

The examples in this section assume that the ruler units have been set to inches.

To resize the image so that it is four inches wide by four inches high, use the document's `resize (Resize/resize)` command.

AS:
```
resize image current document width 4 height 4
```

VB:
```
docRef.ResizeImage 4,4
```

JS:
```
docRef.resizeImage( 4,4 );
```

Resizing the canvas is done similarly.

AS:
```
resize canvas current document width 4 height 4
```

VB:
```
docRef.ResizeCanvas 4,4
```

JS:
```
docRef.resizeCanvas( 4,4 );
```

To trim the excess space from a document, use the `trim (Trim/trim)` command. The example below will trim the top and bottom of the document.

AS:
```
trim current document basing trim on top left pixel ¬
    with top trim and bottom trim without left trim and right trim
```

VB:
```
docRef.Trim Type:=psTopLeftPixel, Top:=True, Left:=False, _
    Bottom:=True, Right:=False
```

JS:
```
docRef.trim(TrimType.TOPLEFT, true, false, true, false);
```

NOTE: The crop command uses unit values. The examples below assume that the ruler unit is set to pixels.

AS:
```
crop current document bounds {10, 20, 40, 50} angle 45 ¬
    resolution 72 width 20 height 20
```

VB:
```
docRef.Crop Array(10,20,40,50), Angle:=45, Width:=20, _
    Height:=20, Resolution:=72
```

JS:

```
docRef.crop (new Array(10,20,40,50), 45, 20, 20, 72);
```

To flip the canvas horizontally:

AS: `flip canvas current document direction horizontal`

VB: `docRef.FlipCanvas psHorizontal`

JS: `docRef.flipCanvas(Direction.HORIZONTAL);`

3.10 Layer objects

Photoshop has 2 types of layers: an `art layer` that can contain image contents and a `layer set` that can contain zero or more art layers. Scripts, likewise, have two types of layers: `art layer` and `layer set`.

Both types of layers have common properties such as “visible.” The common attributes are placed in a general “layer” class that both “art layer” and “layer set” inherit from.

When you create a layer you must specify whether you are creating an art layer or a layer set. The following examples show how to create an art layer filled with red at the beginning of the current document

AS:

```
tell application "Adobe Photoshop CS"
    make new art layer at beginning of current document ¬
       with properties {name:"MyBlendLayer", blend mode:normal}
    select all current document
    fill selection of current document with contents ¬
       {class:RGB color, red:255, green:0, blue:0}
end tell
```

VB:

```
Dim appRef As Photoshop.Application
Set appRef = CreateObject("Photoshop.Application")

' Create a new art layer at the beginning of the current document
Dim docRef As Photoshop.Document
Dim layerObj As Photoshop.ArtLayer
Set docRef = appRef.ActiveDocument
Set layerObj = appRef.ActiveDocument.ArtLayers.Add
layerObj.Name = "MyBlendLayer"
layerObj.BlendMode = psNormalBlend

' Select all so we can apply a fill to the selection
appRef.ActiveDocument.Selection.SelectAll

' Create a color to be used with the fill command
Dim colorObj As Photoshop.SolidColor
Set colorObj = CreateObject("Photoshop.SolidColor")
colorObj.RGB.Red = 255
colorObj.RGB.Green = 100
colorObj.RGB.Blue = 0

' Now apply fill to the current selection
appRef.ActiveDocument.Selection.Fill colorObj
```

JS:

```
// Create a new art layer at the beginning of the current document
var layerRef = app.activeDocument.artLayers.add();
layerRef.name = "MyBlendLayer";
layerRef.blendMode = BlendMode.NORMAL;

// Select all so we can apply a fill to the selection
app.activeDocument.selection.selectAll;

// Create a color to be used with the fill command
var colorRef = new SolidColor;
colorRef.rgb.red = 255;
colorRef.rgb.green = 100;
colorRef.rgb.blue = 0;

// Now apply fill to the current selection
app.activeDocument.selection.fill(colorRef);
```

The following examples show how to create a layer set after the first layer in the current document:

AS:

```
tell application "Adobe Photoshop CS"
    make new layer set after layer 1 of current document
end tell
```

VB:

```
Dim appRef As Photoshop.Application
Set appRef = CreateObject("Photoshop.Application")

' Get a reference to the first layer in the document
Dim layerRef As Photoshop.Layer
Set layerRef = appRef.ActiveDocument.Layers(1)

' Create a new LayerSet (it will be created at the beginning of the
' document)
Dim newLayerSetRef As Photoshop.LayerSet
Set newLayerSetRef = appRef.ActiveDocument.LayerSets.Add


' Move the new layer to after the first layer
newLayerSetRef.Move layerRef, psPlaceAfter
```

JS:

```
// Get a reference to the first layer in the document
var layerRef = app.activeDocument.layers[0];

// Create a new LayerSet (it will be created at the beginning of the
// document)
var newLayerSetRef = app.activeDocument.layerSets.add();

// Move the new layer to after the first layer
newLayerSetRef.move(layerRef, ElementPlacement.PLACEAFTER);
```

An existing art layer can also be changed to a text layer if the existing layer is empty. Conversely you can change a text layer to a normal layer. When you do this the text in the layer is rasterized.

3.10.1 Setting the Active layer

Before attempting to manipulate a layer you must first select it. You can do this by setting the `current layer (ActiveLayer/activeLayer)` to the one you want to manipulate.

AS:

```
set current layer of current document to layer "Layer 1" of ¬
    current document
```

VB:

```
docRef.ActiveLayer = docRef.Layers("Layer 1")
```

JS:

```
docRef.activeLayer = docRef.layers["Layer 1"];
```

3.10.2 Layer sets

Existing layers can be moved into layer sets. The following examples show how to create a layer set, duplicate an existing layer, and move the duplicate layer into the layer set.

AS:

```
set current document to document "My Document"
set layerSetRef to make new layer set at end of current document
set newLayer to duplicate layer "Layer 1" of current document¬
    to end of current document
move newLayer to end of layerSetRef
```

In AppleScript, you can also duplicate a layer directly into the destination layer set.

```
set current document to document "My Document"
set layerSetRef to make new layer set at end of current document
duplicate layer "Layer 1" of current document to end of layerSetRef
```

In Visual Basic and JavaScript you'll have to duplicate and place the layer. Here's how:

VB:

```
Set layerSetRef = docRef.LayerSets.Add
Set layerRef = docRef.ArtLayers(1).Duplicate
                              layerSetRef,psPlaceAtEnd
layerRef.MoveToEnd layerSetRef
```

JS:

```
var layerSetRef = docRef.layerSets.add();
var layerRef = docRef.artLayers[0].duplicate(layerSetRef,
                              ElementPlacement.PLACEATEND);
layerRef.moveToEnd (layerSetRef);
```

3.10.3 Linking layers

Scripting also supports linking and unlinking layers. You may want to link layers together so that moving or transforming them can be done with one statement. To link layers together, do the following:

AS:

```
make new art layer in current document with properties {name:"L1"}
make new art layer in current document with properties {name:"L2"}
link art layer "L1" of current document with art layer "L2" of ¬
    current document
```

VB:

```
Set layer1Ref = docRef.ArtLayers.Add()
Set layer2Ref = docRef.ArtLayers.Add()
layer1Ref.Link layer2Ref.Layer
```

JS:

```
var layerRef1 = docRef.artLayers.add();
var layerRef2 = docRef.artLayers.add();
layerRef1.link(layerRef2);
```

3.10.4 Applying styles to layers

Styles can be applied to layers from your scripts. The styles correspond directly to the styles in the Photoshop Styles palette and are referenced by their literal string name. Here is an example of how to set a layer style to the layer named “L1.”

NOTE: The layer styles name is case sensitive.

AS:

```
apply layer style art layer "L1" of current document using ¬
    "Puzzle (Image)"
```

VB:

```
docRef.ArtLayers("L1").ApplyStyle "Puzzle (Image)"
```

JS:

```
docRef.artLayers["L1"].applyStyle("Puzzle (Image)");
```

3.10.5 Rotating layers

Use the `rotate (Rotate/rotate)` command on the layer to rotate the entire layer. Positive integers rotate the layer clockwise. Negative integers rotate it counterclockwise.

AS:

```
rotate current layer of current document angle 45.0
```

VB:

```
docRef.ActiveLayer.Rotate 45.0
```

JS:

```
docRef.activeLayer.rotate(45.0);
```

3.11 Text item object

In Photoshop, the `Text` object is a property of the art layer. To create a new text layer, you must create a new art layer and then set the art layer's `kind (Kind/kind)` property to `text layer (psTextLayer/ LayerKind.TEXT)`.

NOTE: You may want to refer back to chapter 1 for a quick explanation of the multi-language format used above. See “Conventions in this guide” on page 1.

By changing an art layer's kind, you can also convert an existing layer to text as long as the layer is empty. For example, to create a new text layer, write:

AS:

```
make new art layer in current document with properties ¬
    { kind: text layer }
```

VB:

```
set newLayerRef = docRef.ArtLayers.Add()
newLayerRef.Kind = psTextLayer
```

JS:

```
var newLayerRef = docRef.artLayers.add();
newLayerRef.kind = LayerKind.TEXT;
```

To check if an existing layer is a text layer, you must compare the layer's `kind` to `text layer (psTextLayer/LayerKind.TEXT)`.

AS:

```
if (kind of layerRef is text layer) then
```

VB:

```
If layerRef.Kind = psTextLayer Then
```

JS:

```
if (newLayerRef.kind == LayerKind.TEXT)
```

The art layer class has a `text object (TextItem/textItem)` property which is only valid when the art layer's kind is `text` layer. You can use this property to make modifications to your `text layer` such as setting its contents, changing its size, and controlling the different effects that can be applied to text. For example, to set the justification of your text to right justification, you write:

AS:

```
set justification of text object of art layer "my text" of ¬
    current document to right
```

VB:

```
docRef.ArtLayers("my text").TextItem.Justification = psRight
```

JS:

```
docRef.artLayers["my text"].textItem.justification =
    Justification.RIGHT;
```

IMPORTANT: *The* `text item` *object has a* `kind` *property, which can be set to either* `point text (psPointText/TextType.POINTTEXT)` *or* `paragraph text (psParagraphText/TextType.PARAGRAPHTEXT)`*. When a new* `text item` *is created, its* `kind` *property is automatically set to* `point text`.

The `text item` *properties* `height, width` *and* `leading` *are only valid when the text item's kind property is set to* `paragraph text`.

3.11.1 Setting the contents of the text item

To set the contents of a text item in AppleScript you would write:

```
set contents of text object of art layer "Layer 1" of ¬
    current document to "Hello"
```

If you use a text item object reference to set the contents you will need to write:

```
set contents of contents of textItemRef to "Hello"
```

The second “contents of” is needed because “contents” is a keyword which tells AppleScript to operate on the contents of the variable, rather than on the object to which it may refer. This means that AppleScript sees the above line as:

```
set text object of art layer 1 of document "Untitled-1" ¬
    to "Hello"
```

To set the contents using references in VB and JS, write the following:

VB:

```
textLayerRef.TextItem.Contents = "Hello"
```

JS:

```
textLayerRef.textItem.contents = "Hello";
```

3.11.2 Setting text stroke colors

Setting the stroke color in AppleScript is a bit different then setting it in Visual Basic or JavaScript. To set the stroke color in AppleScript, use one of the color classes: CMYK color, gray color, HSB color, Lab color, or RGB color.

To set it in Visual Basic or JavaScript, you must first create a `SolidColor` object and appropriately assign one of the color classes to it. The following examples show how to set the stroke color of a text item-object to a CMYK color.

See section 3.15, “Color objects” on page 91 for more information on working with colors.

AS:

```
set stroke color of textItemRef to {class:CMYK color, cyan:20,¬
    magenta:50, yellow:30, black:0}
```

VB:

```
Set newColor = CreateObject ("Photoshop.SolidColor")
newColor.CMYK.Cyan = 20
newColor.CMYK.Magenta = 100
newColor.CMYK.Yellow = 30
newColor.CMYK.Black = 0
textLayerRef.TextItem.Color = newColor
```

JS:

```
var newColor = new SolidColor();
newColor.cmyk.cyan = 20;
newColor.cmyk.magenta = 100;
newColor.cmyk.yellow = 30;
newColor.cmyk.black = 0;
textLayerRef.color = newColor;
```

3.11.3 Setting fonts

To set the font of your text item object, set the text item's `font` property. The font names that you can use are the PostScript® names for the fonts. The PostScript names are not the names that are displayed in Photoshop's character palette. The steps below show how to find a PostScript font name.

1. Using the Photoshop user interface, create a new Photoshop document.
2. Create a new text layer and add some text to it.
3. Select the text you created in step 2.
4. Select the desired font from the Font pull down menu (for example, "Arial")

5. Create a script to get the font name of the text. An example JavaScript is below:

```
var textLayer = activeDocument.artLayers[0];
if (textLayer.kind == LayerKind.TEXT)
{
     alert(textLayer.textItem.font);
}
```

6. The name that is displayed in the alert dialog is the PostScript name of the font. Use this name to set the font of your text For example, the above script returned the name "ArialMT." The examples below show how to set this font:

AS: `set font of textItemRef to "ArialMT"`

VB: `textLayer.TextItem.Font = "ArialMT"`

JS: `textLayer.textItem.font = "ArialMT";`

3.11.4 Warping text

Warping is another common effect that can be applied to text. To warp a text item-object, set the object's `warp style (WarpStyle/warpStyle)` property. The style to set it to is an enumeration.

AS:

```
set warp style of textItemRef to flag
```

VB:

```
textLayerRef.TextItem.WarpStyle = psFlag
```

JS:

```
textLayerRef.textItem.warpStyle = WarpStyle.FLAG;
```

3.12 Selections

There are instances where you will want to write scripts that only act on the current selection. If you are writing a script that depends on a selection, be sure to set the selection yourself, as you cannot test for a non-existent selection. When creating new selections, you can add to, replace, or subtract from a selection.

For example, you may apply effects to a selection or copy the current selection to the clipboard. But remember that you may have to set the active layer before acting on the selection. Here's how:

AS:

```
set current layer of current document to layer "Layer 1" of ¬
    current document
```

VB:

```
docRef.ActiveLayer = docRef.Layers("Layer 1")
```

JS:

```
docRef.activeLayer = docRef.layers["Layer 1"];
```

See section 3.10.1, "Setting the Active layer" on page 75 for more information.

3.12.1 Defining selections

To create a new selection, use the `select` method with a type of `replaced` (`psReplaceSelection/SelectionType.REPLACED`). The other selection types are `diminished, extended` and `intersected`.

The `diminished` type shrinks the current selection, the `extended` selection type expands the current selection, and the `intersected` type finds the intersection of the current selection and the new selection and replace the current selection with the intersection of the two.

If there is no intersection between the selections, the new selection will be empty. If there is no current selection, the new selection will be the newly specified selection.

Here are examples of how to replace the current selection:

AS:

```
select current document region {{ 5, 5}, {5, 100}, ¬
    { 80, 100}, { 80, 5}} combination type replaced
```

VB:

```
Dim appRef As New Photoshop.Application

'remember unit settings; and set to values expected by this script
Dim originalRulerUnits As Photoshop.PsUnits
originalRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = psPixels

'get selection and replace it
Dim docRef As Photoshop.Document
Set docRef = appRef.ActiveDocument
docRef.Selection.Select Array(Array(50, 60), Array(150, 60), _
    Array(150, 120), Array(50, 120)), Type:=psReplaceSelection

'restore unit setting
appRef.Preferences.RulerUnits = originalRulerUnits
```

JS:

```
// remember unit settings; and set to values expected by this
// script
var originalRulerUnits = app.preferences.rulerUnits;
app.preferences.rulerUnits = Units.PIXELS;

//get selection and replace it;
app.activeDocument.selection.select (new Array(new Array(60, 10),
    new Array(100, 10), new Array(100, 100), new Array(60, 100)),
          SelectionType.REPLACE);

// restore unit setting
app.preferences.rulerUnits = originalRulerUnits;
```

3.12.2 Stroking the selection border

The following examples show how to stroke the boundaries around the current selection and set the stroke color and width.

AS:

```
stroke selection of current document using color ¬
    {class:CMYK color,cyan:20, magenta:50, yellow:30, black:0}¬
    width 5 location inside blend mode vivid light opacity 75 ¬
    without preserving transparency
```

VB:

```
selRef.Stroke strokeColor, Width:=5, Location:=psInsideStroke, _
    mode:=psVividLightBlend, Opacity:=75, _
          PreserveTransparency:=False
```

JS:

```
app.activeDocument.selection.stroke (strokeColor, 2,
    StrokeLocation.OUTSIDE, ColorBlendMode.VIVIDLIGHT, 75,
          false);
```

IMPORTANT: *The transparency parameter cannot be used for background layers.*

3.12.3 Inverting selections

When you invert a selection, you are masking the selection so you can work on the rest of the document, layer or channel while protecting the selection. Here's how to invert the current selection:

AS: `invert selection of current document`

VB: `selRef.Invert`

JS: `selRef.invert();`

3.12.4 Expand, contract and feather selections

These three commands are used to change the size of the selection. The values are passed in ruler units, the values of which are stored in Photoshop preferences and can be changed by your scripts. Feathering a selection will smooth its corners by the specified number of units while expand and contract will grow and shrink the selection.

If your ruler units are set to pixels, then the following examples will expand, contract and feather by five pixels. See section 3.6.3, "Changing ruler and type units" on page 61 for examples of how to change ruler units.

AS:

```
expand selection of current document by pixels 5
contract selection of current document by pixels 5
feather selection of current document by pixels 5
```

VB:

```
Dim appRef As Photoshop.Application
Set appRef = CreateObject("Photoshop.Application")

'remember unit settings; and set to pixels
Dim originalRulerUnits As Photoshop.PsUnits
originalRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = psPixels

Dim selRef As Photoshop.Selection
Set selRef = appRef.ActiveDocument.Selection

selRef.Expand 5
selRef.Contract 5
selRef.Feather 5

'Rem restore unit setting
appRef.Preferences.RulerUnits = originalRulerUnits
```

JS:

```
// remember unit settings; and set to pixels
var originalRulerUnits = app.preferences.rulerUnits;
app.preferences.rulerUnits = Units.PIXELS;

var selRef = app.activeDocument.selection
selRef.expand( 5 );
selRef.contract( 5 );
selRef.feather( 5 );

// restore unit setting
app.preferences.rulerUnits = originalRulerUnits;
```

3.12.5 Filling a selection

You can fill a selection either will a color or a history state. To fill with a color:

AS:

```
fill selection of current document with contents ¬
    {class: RGB color, red:255, green:0, blue:0} blend mode ¬
       vivid light opacity 25 without preserving transparency
```

VB:

```
Set fillColor = CreateObject("Photoshop.SolidColor")
fillColor.RGB.Red = 255
fillColor.RGB.Green = 0
fillColor.RGB.Blue = 0
selRef.Fill fillColor mode:=psVividLightBlend, _
    Opacity:=25, PreserveTransparency:=False
```

JS:

```
var fillColor = new SolidColor();
fillColor.rgb.red = 255;
fillColor.rgb.green = 0;
fillColor.rgb.blue = 0;
app.activeDocument.selection.fill( fillColor,
ColorBlendMode.VIVIDLIGHT,
    25, false);
```

To fill the current selection with the 10th item in the history state, you would write:

AS:

```
fill selection of current document with contents history state 10 ¬
    of current document
```

VB:

```
selRef.Fill docRef.HistoryStates(10)
```

JS:

```
selRef.fill(app.activeDocument.historyStates[9]);
```

3.12.6 Rotating selections

You can rotate either the contents of a selection or the selection boundary itself. The first set of examples below shows how to rotate an existing selection 45 degrees.

AS:

```
rotate selection of current document angle 45
```

VB:

```
docRef.Selection.Rotate(45);
```

JS:

```
app.activeDocument.selection.rotate(45);
```

To rotate the boundary of an existing selection:

AS:

```
rotate boundary selection of current document angle 45
```

VB:

```
docRef.Selection.RotateBoundary(45);
```

JS:

```
app.activeDocument.selection.rotateBoundary(45);
```

3.12.7 Loading and storing selections

Photoshop exposes the functionality to store and load selections. Selections get stored into channels and loaded from channels.The following examples store the current selection into a channel named “My Channel” and extend the selection with any selection that is currently in that channel.

AS:

```
store selection of current document into channel "My Channel" of ¬
    current document combination type extended
```

VB:

```
selRef.Store docRef.Channels("My Channel"), psExtendSelection
```

JS:

```
selRef.store(docRef.channels["My Channel"], SelectionType.EXTEND);
```

To restore a selection that has been saved to a selection, use the load (Load/load) method as shown below.

AS:

```
load selection of current document from channel "My Channel" of ¬
    current document combination type extended
```

VB:

```
selRef.Load docRef.Channels("My Channel"), psExtendSelection
```

JS:

```
selRef.load (docRef.channels["My Channel"], SelectionType.EXTEND);
```

See section 3.17, “Clipboard interaction” on page 96 for examples on how to copy, cut and paste selections.

3.13 Working with Filters

To apply a filter, use the layer's `filter` command for AppleScript or the `ApplyXXX/applyXXX` methods for Visual Basic and JavaScript. The following examples apply the Gaussian blur filter to the active layer.

AS:

```
filter current layer of current document using Gaussian blur ¬
    with options { radius: 5 }
```

VB:

```
docRef.ActiveLayer.ApplyGaussianBlur 5
```

JS:

```
docRef.activeLayer.applyGaussianBlur(5);
```

3.13.1 Selecting channel(s) to filter

When applying filters, keep in mind they affect the selected channels of a visible layer. This means that prior to running a filter, you may have to set the active channels. Since more than one channel can be active at a time, you must provide an array of channels when setting a channel. The code below demonstrates how to set the active channels to the channels named “Red” and “Blue.”

AS:

```
set current channels of current document to { channel "Red" of ¬
    current document, channel "Blue" of current document }
```

VB:

```
Dim theChannels As Variant
theChannels = Array(docRef.Channels("Red"), docRef.Channels("Blue"))
docRef.ActiveChannels = theChannels
```

JS:

```
theChannels = new Array(docRef.channels["Red"],
docRef.channels["Blue"]);
docRef.activeChannels = theChannels;
```

Or you can easily select all component channels by using the “component channel” property on the document:

AS:

```
set current channels of current document to component channels ¬
    of current document
```

VB:

```
appRef.ActiveDocument.ActiveChannels= _
    appRef.ActiveDocument.ComponentChannels
```

JS:

```
app.activeDocument.activeChannels =
    activeDocument.componentChannels;
```

3.13.2 Other filters

If the filter type that you want to use on your layer is not part of the scripting interface, you can use the Action Manager from a JavaScript to run a filter. If you are using AppleScript, Visual Basic or VBScript, you can still run a JavaScript from your script.

3.14 Channel object

The Channel object `(Channel/channel)` gives you access to much of the available functionality on Photoshop channels. You can create, delete and duplicate channels or retrieve a channel's histogram and change its kind or change the current channel selection.

3.14.1 Channel types

In addition to the component channels, Photoshop lets you to create additional channels. You can create a “spot color channel”, a “masked area channel” and a “selected area channel.”

It’s important to keep the different types of channels in mind when writing scripts that work on them.

If you have an RGB document you automatically get a red, blue and a green channel. These kinds of channels are related to the document mode and are called "component channels."

A Channel has a `kind` property you can use to get and set the type of the channel. Possible values are: `component channel`, `masked area channel`, `selected area channel` and `spot color channel`.

You cannot change the kind of a component channel. But you could change a "masked area channel" to be "selected area channel" by saying:

AS: `set kind of myChannel to selected area channel`

VB: `channelRef.kind = psSelectedAreaAlphaChannel`

JS: `channelRef.kind = ChannelType.SELECTEDAREA;`

NOTE: You cannot use book colors or convert document mode to duo-tone.

3.14.2 Setting the active channel

Because more than one channel can be active at a time, when setting a channel, you must provide a channel array. The sample below demonstrates how to set the active channels to the first and third channel.

AS:

```
set current channels of current document to ¬
    { channel 1 of current document, channel 3 of current document
}
```

VB:

```
Dim theChannels As Variant
theChannels = Array(docRef.Channels(1), docRef.Channels(3))
docRef.ActiveChannels = theChannels
```

JS:

```
theChannels = new Array(docRef.channels[0], docRef.channels[2]);
docRef.activeChannels = theChannels;
```

Deleting a component will change the document to a multi-channel document.

3.14.3 Creating new channels

You can create three different types of channels from a script. These types are:

- `masked area channel (psMaskedAreaAlphaChannel, ChannelType.MASKEDAREA)`
- `selected area channel (psSelectedAreaAlphaChannel, ChannelType.SELECTEDAREA)`

- `spot color channel (psSpotColorChannel, ChannelType.SPOTCOLOR)`.

The examples below show how to create a new masked area channel.

AS:

```
make new channel in current document with properties ¬
    {kind:masked area channel }
```

VB:

```
Set channelRef = docRef.Channels.Add
channelRef.Kind = psMaskedAreaChannel
```

JS:

```
var channelRef = docRef.channels.add();
channelRef.kind = ChannelType.MASKEDAREA;
```

3.15 Color objects

From scripting you can use the same range of colors that are available from the Photoshop user interface. Each has its own set of properties, which are specific to the color. For example, the RGB color class contains three properties — red, blue and green.

3.15.1 Setting a Color

Here's how to set the foreground color to a CMYK color in AppleScript.

```
set foreground color to {class:CMYK color, cyan:20.0, ¬
    magenta:90.0, yellow:50.0, black:50.0}
```

Because you can use any color model, you could also write the following to set the foreground to an RGB color.

```
set foreground color to { class:RGB color, red:80.0, green:120.0,¬
    blue:57.0 }
```

In Visual Basic and JavaScript, the `SolidColor` object handles all colors. To set the foreground color you should create a `SolidColor` object, set its color model by assigning the color model values and then set the foreground color to the solid color. Here's how:

VB:

```
solidColor = CreateObject("Photoshop.SolidColor")
appRef.ForegroundColor = solidColor
```

JS:

```
var solidColor = new SolidColor();
foregroundColor = solidColor;
```

SolidColor class

Visual Basic and JavaScript have an additional class called the `SolidColor` class. This class contains a property for each color model. To use this object, first create an instance of a `SolidColor` object, then set its appropriate color model properties. Once a color model has been assigned to a `SolidColor` object, the `SolidColor` object cannot be reassigned to a different color model. Below are examples for creating a `SolidColor` object and set its CMYK property.

VB:

```
Dim solidColor As Photoshop.SolidColor
Set solidColor = CreateObject("Photoshop.SolidColor")
solidColor. CMYK.Cyan = 20
solidColor.CMYK.Magenta = 90
solidColor.CMYK.Yellow = 50
solidColor.CMYK.Black = 50
```

JS:

```
var solidColor = new SolidColor();
solidColor.cmyk.cyan = 20;
solidColor.cmyk.magenta = 90;
solidColor.cmyk.yellow = 50;
solidColor.cmyk.black = 50;
```

Hex values

An RGB color can also be represented as a hex value. The hexadecimal value is used to represent the three colors of the RGB model. The hexadecimal value contains three pairs of numbers which when read from left to right, represent the red, blue and green colors.

In AppleScript, the hex value is represented by the `hex value` string property in class `RGB hex color`, and you use the `convert color` command described below to retrieve the hex value.

In Visual Basic and JavaScript, the RGBColor object has a string property called `HexValue/hexValue`.

3.15.2 Getting and converting colors

Here's how to get the foreground color in AppleScript.

```
get foreground color
```

This may return an RGB color and in some cases you may want the CMYK equivalent. To convert an RGB color to CMYK in AppleScript you would write:

```
convert color foreground color to CMYK
```

VB/JS:

The foreground color returns a SolidColor object. You should use its `model` property to determine the color model.

```
If (someColor.model = ColorModel.RGB) Then
    alert("It's an RGB color")
End If
```

You can also ask the `SolidColor` object to convert its color to any of the supported models. For example, writing:

```
someColor.cmyk
```

will return a CMYKColor object representing the CMYK version of the color in `someColor` regardless of the color model of someColor.

The examples below show how to convert the foreground color to a Lab color.

AS:

```
-- Convert foreground application color to Lab
set myLabColor to convert color foreground color to Lab
```

VB:

```
' Get the foreground color as Lab
Dim myLabColor As Photoshop.LabColor
Set myLabColor = appRef.ForegroundColor.Lab
```

JS:

```
// Get the Lab color from the foreground color.
var myLabColor = foregroundColor.lab;
```

3.15.3 Comparing Colors

Using the `equal colors (IsEqual/isEqual)` commands, you can easily compare colors. These methods will return `true` if the colors are visually equal to each other and `false` otherwise. The examples below compare the foreground color to the background color.

AS:
```
if equal colors foreground color with background color then
```

VB:
```
If (appRef.ForegroundColor.IsEqual(appRef.BackgroundColor)) Then
```

JS:
```
if (app.foregroundColor.isEqual(backgroundColor))
```

3.15.4 Getting a Web Safe Color

To convert a color to a web safe color use the `web safe color` command on AppleScript and the `NearestWebColor/nearestWebColor` property on the `SolidColor` object for Visual Basic and JavaScript. The web safe color returned is an RGB color.

AS:

```
set myWebSafeColor to web safe color for foreground color
```

VB:

```
Dim myWebSafeColor As Photoshop.RGBColor
Set myWebSafeColor = appRef.ForegroundColor.NearestWebColor
```

JS:

```
var webSafeColor = new RGBColor();
webSafeColor = app.foregroundColor.nearestWebColor;
```

3.16 History object

Photoshop keeps a history of the actions that affect the appearance of documents. Each entry in the Photoshop History palette is considered a “History State.” These states are accessable from document object and can be used to reset the document to a previous state. A history state can also be used to fill a selection.

To set your document back to a particular state, set the document's current history state:

AS:

```
set current history state of current document to history state 1 ¬
    of current document
```

VB:

```
docRef.ActiveHistoryState = docRef.HistoryStates(1)
```

JS:

```
docRef.activeHistoryState = docRef.historyStates[0];
```

The code above sets the current history state to the top history state that is in the History palette. Using history states in this fashion gives you the ability to undo the actions that were taken to modify the document.

The example below saves the current state, applies a filter, and then reverts back to the saved history state.

AS:

```
set savedState to current history state of current document
filter current document using motion blur with options ¬
    {angle:20, radius: 20}
set current history state of current document to savedState
```

VB:

```
Set savedState = docRef.ActiveHistoryState
docRef.ApplyMotionBlur 20, 20
docRef.ActiveHistoryState = savedState
```

JS:

```
savedState = docRef.activeHistoryState;
docRef.applyMotionBlur( 20, 20 );
docRef.activeHistoryState = savedState;
```

IMPORTANT: *Reverting back to a previous history state does not remove any latter states from the history collection. Use the* `Purge` *command to remove latter states from the history collection as shown below:*

AS: `purge history caches`

VB: `appRef.Purge( psHistoryCaches)`

JS: `app.purge( PurgeTarget.HISTORYCACHES );`

3.16.1 Filling a selection with a history state

A history state can also be used to fill a selection. See section 3.12, “Selections” on page 81 for more information on working with selections.

3.17 Clipboard interaction

The clipboard commands in Photoshop operate on layers and selections. The commands can be used to operate on a single document, or to move information between documents.

NOTE: On Mac OS, Photoshop must be the front-most application when executing these commands. You must activate the application before executing any clipboard commands.

3.17.1 Copy

The example below shows how to copy the contents of art layer 2 to the clipboard.

AS:

```
activate
select all of current document
copy art layer 2 of current document
```

NOTE: In AppleScript, you must select the entire layer before performing the copy.

VB:

```
docRef.ArtLayers(2).Copy
```

JS:

```
docRef.artLayers[1].copy();
```

3.17.2 Copy merged

You can also perform a merged copy. This will make a copy of all visible layers in the selected area.

In AppleScript, use the copy merged command.

AS:

```
activate
select all of current document
copy merged selection of current document
```

In VB and JS, pass `true` for the Merged parameter of the Copy methods.

VB:

```
docRef.Selection.Copy True
```

JS:

```
docRef.selection.copy(true);
```

3.17.3 Cut

The Cut command operates on both art layers and selections, so you can cut an entire art layer or only the selection of a visible layer. The Cut method follows the same rules as the copy command.

AS:

```
activate
cut selection of current layer of current document
```

VB:

```
docRef.Selection.Cut
```

JS:

```
docRef.selection.cut();
```

3.17.4 Paste

The paste command can be used on any open document, and operates on the current document. You must make the paste command's target document the current document before using the command. A new layer is created by the paste command, and a reference to it is returned. If there is no selection in the target document, the contents of the clipboard are pasted into the geometric center of the document.

Here's how to paste into a document named "Doc2":

AS:

```
activate
set current document to document "Doc2"
set newLayerRef to paste
```

In Visual Basic and JavaScript the paste command is defined on the Document object.

VB:

```
appRef.ActiveDocument = appRef.Documents("Doc2")
Set newLayerRef = docRef.Paste
```

JS:

```
activeDocument = documents["Doc2"];
var newLayerRef = docRef.paste();
```

3.17.5 Paste into command

The paste into command allows you to paste the contents of the clipboard into the selection in a document. The destination selection border is then converted into a layer mask. You must make the paste command's target document the current document before using the command.

AS:

```
activate
set newLayerRef to paste with clipping to selection
```

VB:

```
Set newLayerRef = docRef.Paste (True)
```

JS:

```
newLayerRef = docRef.paste( true );
```

Index

H

I

J

L

N

O

P

R

S

T

U

V

W

X