## Contents

1. **Introduction** ................................................................................................................. 6
   - About this Manual ........................................................................................................ 6
   - Conventions in this Guide ........................................................................................... 6

2. **Photoshop Scripting Basics** ...................................................................................... 8
   - Scripting Overview ....................................................................................................... 8
     - Why use scripts instead of actions? ........................................................................... 8
   - Scripting Support in Photoshop .................................................................................. 9
     - JavaScript support ..................................................................................................... 9
     - Executing other scripts ............................................................................................ 10
   - Startup scripts ............................................................................................................ 10
   - Executing JavaScripts from AS or VBS ..................................................................... 10
   - Photoshop Object Model ........................................................................................... 11
     - Containment hierarchy ............................................................................................. 11
       - Application and document classes ....................................................................... 12
       - Layer classes .......................................................................................................... 12
       - Layer Comp class .................................................................................................. 13
       - Channel class ....................................................................................................... 13
       - Selection class ...................................................................................................... 13
       - History State class ................................................................................................. 13
       - Document Info class .............................................................................................. 13
       - Path Item, Sub Path Item, and Path Point classes ................................................ 14
       - Preferences class .................................................................................................... 14
       - Notifier class ......................................................................................................... 14
       - Count Item class .................................................................................................... 14
       - Color Sampler class ............................................................................................... 14
       - Measurement Scale class ....................................................................................... 14
     - The containment hierarchy and the Photoshop user interface ............................... 14
     - Additional objects .................................................................................................... 16
     - Constants .................................................................................................................. 16
   - Creating a sample Hello World script ........................................................................... 17
   - Creating and running an AppleScript .......................................................................... 18
   - Creating and running a VBScript ................................................................................ 19
   - Creating and running a JavaScript ............................................................................... 19

3. **Scripting Photoshop** .................................................................................................. 21
   - Viewing Photoshop Objects, Commands, and Methods .............................................. 21
     - Viewing Photoshop's AppleScript dictionary ......................................................... 21
     - Viewing Photoshop's type library (VBS) ................................................................. 22
   - Targeting and Referencing the Application Object ....................................................... 22
   - Creating New Objects in a Script ................................................................................ 23
   - Setting the Active Object ............................................................................................ 25
     - Setting the active document .................................................................................... 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting the active layer</td>
<td>27</td>
</tr>
<tr>
<td>Setting the active channels</td>
<td>28</td>
</tr>
<tr>
<td>Opening a Document</td>
<td>28</td>
</tr>
<tr>
<td>Opening a file with default file format</td>
<td>28</td>
</tr>
<tr>
<td>Specifying file formats to open</td>
<td>29</td>
</tr>
<tr>
<td>Saving a Document</td>
<td>31</td>
</tr>
<tr>
<td>Setting Application Preferences</td>
<td>32</td>
</tr>
<tr>
<td>Allowing or Preventing Dialogs</td>
<td>33</td>
</tr>
<tr>
<td>Working with the Photoshop Object Model</td>
<td>33</td>
</tr>
<tr>
<td>Using the Application object</td>
<td>34</td>
</tr>
<tr>
<td>Using the Document object</td>
<td>34</td>
</tr>
<tr>
<td>Manipulating a document object</td>
<td>35</td>
</tr>
<tr>
<td>Working with layer objects</td>
<td>36</td>
</tr>
<tr>
<td>Creating an ArtLayer object</td>
<td>37</td>
</tr>
<tr>
<td>Creating a Layer Set object</td>
<td>38</td>
</tr>
<tr>
<td>Referencing ArtLayer objects</td>
<td>38</td>
</tr>
<tr>
<td>Working with Layer Set objects</td>
<td>39</td>
</tr>
<tr>
<td>Linking layer objects</td>
<td>40</td>
</tr>
<tr>
<td>Applying styles to layers</td>
<td>40</td>
</tr>
<tr>
<td>Using the Text Item object</td>
<td>41</td>
</tr>
<tr>
<td>Determining a layer’s kind</td>
<td>41</td>
</tr>
<tr>
<td>Adding and manipulating text in a Text Item object</td>
<td>42</td>
</tr>
<tr>
<td>Working with Selection objects</td>
<td>42</td>
</tr>
<tr>
<td>Creating and defining a selection</td>
<td>43</td>
</tr>
<tr>
<td>Stroking the selection border</td>
<td>44</td>
</tr>
<tr>
<td>Inverting selections</td>
<td>44</td>
</tr>
<tr>
<td>Expanding, contracting, and feathering selections</td>
<td>44</td>
</tr>
<tr>
<td>Filling a selection</td>
<td>45</td>
</tr>
<tr>
<td>Loading and storing selections</td>
<td>45</td>
</tr>
<tr>
<td>Working with Channel objects</td>
<td>46</td>
</tr>
<tr>
<td>Changing channel types</td>
<td>46</td>
</tr>
<tr>
<td>Using the Document Info object</td>
<td>47</td>
</tr>
<tr>
<td>Using history state objects</td>
<td>47</td>
</tr>
<tr>
<td>Using Notifier objects</td>
<td>48</td>
</tr>
<tr>
<td>Using the PathItem object</td>
<td>49</td>
</tr>
<tr>
<td>Working with color objects</td>
<td>51</td>
</tr>
<tr>
<td>Solid color classes</td>
<td>52</td>
</tr>
<tr>
<td>Using hex values for RGB color</td>
<td>52</td>
</tr>
<tr>
<td>Getting and converting colors</td>
<td>52</td>
</tr>
<tr>
<td>Comparing colors</td>
<td>53</td>
</tr>
<tr>
<td>Getting a Web-safe color</td>
<td>53</td>
</tr>
<tr>
<td>Working with Filters</td>
<td>53</td>
</tr>
<tr>
<td>Other filters</td>
<td>54</td>
</tr>
<tr>
<td>Understanding Clipboard Interaction</td>
<td>54</td>
</tr>
<tr>
<td>Using the copy and paste commands</td>
<td>54</td>
</tr>
<tr>
<td>Using the copy merged command/method</td>
<td>55</td>
</tr>
<tr>
<td>Working with Units</td>
<td>56</td>
</tr>
<tr>
<td>Unit values</td>
<td>56</td>
</tr>
<tr>
<td>Special unit value types</td>
<td>56</td>
</tr>
<tr>
<td>Contents</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>AppleScript unit considerations</td>
<td>56</td>
</tr>
<tr>
<td>Using unit values in calculations</td>
<td>57</td>
</tr>
<tr>
<td>Unit value usage</td>
<td>57</td>
</tr>
<tr>
<td>Setting ruler and type units in a script</td>
<td>59</td>
</tr>
<tr>
<td>Sample Workflow Automation JavaScripts</td>
<td>60</td>
</tr>
<tr>
<td>Advanced Scripting</td>
<td>60</td>
</tr>
<tr>
<td>Working with document preferences</td>
<td>61</td>
</tr>
<tr>
<td>Applying color to a text item</td>
<td>64</td>
</tr>
<tr>
<td>Applying a wave filter</td>
<td>66</td>
</tr>
<tr>
<td>Defining the area of a selection object</td>
<td>67</td>
</tr>
<tr>
<td>Applying a MotionBlur filter</td>
<td>71</td>
</tr>
<tr>
<td>4 Action Manager</td>
<td>73</td>
</tr>
<tr>
<td>The ScriptListener Plug-In</td>
<td>73</td>
</tr>
<tr>
<td>Installing ScriptListener</td>
<td>73</td>
</tr>
<tr>
<td>Action Manager Scripting Objects</td>
<td>74</td>
</tr>
<tr>
<td>Recording a Script using ScriptListener</td>
<td>74</td>
</tr>
<tr>
<td>Using the Action Manager from JavaScript</td>
<td>75</td>
</tr>
<tr>
<td>Using the Action Manager from a VBS Script</td>
<td>76</td>
</tr>
<tr>
<td>Running JavaScript-based Action Manager code from VBScript</td>
<td>79</td>
</tr>
<tr>
<td>Running JavaScript-based Action Manager code from AppleScript</td>
<td>80</td>
</tr>
<tr>
<td>Using ScriptListener to find event IDs and class IDs</td>
<td>81</td>
</tr>
<tr>
<td>Index</td>
<td>84</td>
</tr>
</tbody>
</table>
1 Introduction

About this Manual

This manual provides an introduction to scripting Adobe Photoshop CS6 on Mac OS and Windows.

Chapter 1 covers the basic conventions used in this manual.

Chapter 2 covers a brief overview of scripting, how to execute scripts, and the Photoshop object model.

Chapter 3 covers Photoshop-specific objects and components and describes advanced techniques for scripting the Photoshop application. Code examples are provided in three languages:

➤ AppleScript
➤ VBScript
➤ JavaScript™

NOTE: Separate Photoshop scripting reference information is provided for each of these languages through the Scripting Reference Manuals provided with this installation, or through the object browsers available for each language. See “Viewing Photoshop’s AppleScript dictionary” on page 21 and “Viewing Photoshop’s type library (VBS)” on page 22. For information about using the Extend Script Object Model Viewer, see the JavaScript Tools Guide.

Chapter 4 covers the Action Manager, which allows you to write scripts that target Photoshop functionality that is not otherwise accessible in the scripting interface.

NOTE: Please review the README file shipped with Photoshop for late-breaking news, sample scripts, and information about outstanding issues.

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, VBScript and JavaScript. Please note the following shortcut notations:

➤ AS stands for AppleScript
➤ VBS stands for VBScript
➤ JS stands for JavaScript

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

When referring to specific properties and commands, this manual follows the AppleScript naming convention for that property and the VBScript 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 VBScript property and displayDialogs refers to the JavaScript property.

For larger blocks of code, scripting examples are listed on separate lines.

**AS**

layer 1 of layer set 1 of current document

**VBS**

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

**JS**

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

Finally, tables are sometimes used to organize lists of values specific to each scripting language.
This chapter provides an overview of scripting for Photoshop, describes scripting support for the scripting languages AppleScript, VBScript, and JavaScript, how to execute scripts, and covers the Photoshop object model. It provides a simple example of how to write your first Photoshop script.

If you are familiar with scripting or programming languages, you most likely will want to skip much of this chapter. Use the following list to locate information that is most relevant to you.

➤ For more information on the Photoshop object model, see “Photoshop Object Model” on page 11.
➤ For information on selecting a scripting language, refer to the Introduction to Scripting guide.
➤ For examples of scripts created specifically for use with Photoshop, see Chapter 3, “Scripting Photoshop” on page 21.
➤ For detailed information on Photoshop objects and commands, please use the reference information in the three reference manuals provided with this installation: Adobe Photoshop CS6 AppleScript Scripting Reference, Adobe Photoshop CS6 Visual Basic Scripting Reference, and Adobe Photoshop CS6 JavaScript Scripting Reference.

NOTE: You can also view information about the Photoshop objects and commands through the object browsers for each of the three scripting languages. See “Viewing Photoshop Objects, Commands, and Methods” on page 21.

Scripting Overview

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 they can be 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.

If you are new to scripting, you should acquaint yourself with the basic scripting information provided in the Introduction to Scripting manual.

Why use scripts instead of actions?

If you’ve used Photoshop Actions, you’re already familiar with the enormous benefits of automating repetitive tasks. Scripting allows you to extend those benefits by allowing you to add functionality that is not available for Photoshop Actions. For example, you can do the following with scripts and not with actions:

➤ You can add conditional logic, so that the script automatically makes “decisions” based on the current situation. For example, you could write a script that decides which color border to add depending on
the size of the selected area in an image: “If the selected area is smaller than 2 x 4 inches, add a green border; otherwise add a red border.”

➤ A single script can perform actions that involve multiple applications. For example, depending on the scripting language you are using, you could target both Photoshop and another Adobe Creative Suite 6 Application, such as Adobe Illustrator® CS6, in the same script.

➤ You can open, save, and rename files using scripts.

➤ You can copy scripts from one computer to another. If you were using an Action and then switched computers, you’d have to recreate the Action.

➤ Scripts provide more versatility for automatically opening files. When opening a file in an action, you must hard code the file location. In a script, you can use variables for file paths.

**NOTE:** See Photoshop Help for more information on Photoshop Actions.

### Scripting Support in Photoshop

Photoshop supports scripting in three scripting languages: AppleScript, VBScript, and JavaScript. AppleScript and JavaScript run on Mac OS, and JavaScript and VBScript run on Windows. For information about how to choose which scripting language to use, and for additional information about using these languages with Adobe applications, see *Introduction to Scripting*.

See “Creating and running an AppleScript” on page 18, “Creating and running a VBScript” on page 19, and “Creating and running a JavaScript” on page 19.

You can call JavaScript scripts from AppleScript and VBScript scripts. See “Executing JavaScripts from AS or VBS” on page 10.

For a file to be recognized by Photoshop as a valid script file it must have the correct file name extension:

<table>
<thead>
<tr>
<th>Script type</th>
<th>File type</th>
<th>Extension</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppleScript</td>
<td>compiled script OSAS file</td>
<td>.scpt (none)</td>
<td>Mac OS*</td>
</tr>
<tr>
<td>JavaScript ExtendScript</td>
<td>text</td>
<td>.js .jsx</td>
<td>Mac OS &amp; Windows</td>
</tr>
<tr>
<td>VBScript</td>
<td>text</td>
<td>.vbs</td>
<td>Windows</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>executable</td>
<td>.exe</td>
<td>Windows</td>
</tr>
</tbody>
</table>

### JavaScript support

For a JavaScript file to be recognized by Photoshop as a valid script file, it must use either a .js or a .jsx extension. On the Mac OS, there is no difference in the way scripts with the two extensions function. On Windows, if the script files is opened from inside Photoshop, there is no difference between using the .js and .jsx extension. However, if the script is launched by double-clicking on it, a script with the .js extension is interpreted with the Microsoft® JScript engine, and it cannot launch Photoshop. For Windows, using the .jsx extension is preferable, since it interprets the script with the ExtendScript engine.
Scripts written in JavaScript can be accessed from the Adobe Photoshop Scripts menu (File > Scripts), which provides quick and easy access to your JavaScripts. By putting a JavaScript file into the appropriate location on disk, it can be accessed directly from the Photoshop menu.

To install a JavaScript in the Scripts menu, place it in the Scripts folder (Photoshop CS6 /Presets /Scripts). The names of the scripts in the Scripts folder, without the file name extension, are displayed in the Scripts menu. Any number of scripts may be installed in the Scripts menu.

Scripts added to the Scripts folder while Photoshop is running do not appear in the Scripts menu until the next time you launch the application.

All scripts found in the Scripts folder and sub-folders are displayed at the top level of the File > Scripts menu. The addition of sub-folders does not add a hierarchical organization to the Scripts menu.

### Executing other scripts

The Browse item at the end of the Scripts menu (File > Scripts > Browse) allows you to execute scripts that are not installed in the Scripts folder. You can also use Browse to select scripts installed in the Scripts folder after the application was last launched.

Selecting Browse displays a file browser dialog which allows you to select a script file for execution. Only .js or .jsx files are displayed in the browse dialog. When you select a script file, it is executed the same way as an installed script.

### Startup scripts

On startup, Photoshop executes all .jsx files that it finds in the startup folders.

- On Windows, the startup folder for user-defined scripts is:
  
  C:\Program Files\Common Files\Adobe\Startup Scripts CS6\Adobe Photoshop

- On Mac OS, the startup folder for user-defined scripts is:
  
  ~/Library/Application Support/Adobe/Startup Scripts CS6/Adobe Photoshop

If your script is in this main startup folder, it is also executed by all other Adobe Creative Suite 6 applications at startup. If such a script is meant to be executed only by Photoshop, it must include code such as the following:

```javascript
if( BridgeTalk.appName == "photoshop" ) {
   //continue executing script
}
```

For additional details, see the JavaScript Tools Guide.

### Executing JavaScripts from AS or VBS

You can take advantage of JavaScript’s platform-independence by running scripts from AppleScript or VBScript. You can execute either a single JavaScript statement or a complete JavaScript file. For more information, please refer to Introduction to Scripting.
Photoshop Object Model

A document object model (DOM) is an application programming interface (API), which allows you to programmatically access various components of a document (as defined for that application) through a scripting language. For additional information about Adobe object models and the scripting languages that support them, see Introduction to Scripting.

The Photoshop DOM consists of a hierarchical representation of the Photoshop application, the documents used in it, and the components of the documents. The DOM allows you to programmatically access and manipulate the document and its components. For example, through the DOM, you can create a new document, add a layer to an existing document, or change the background color of a layer. Most of the functionality available through the Photoshop user interface is available through the DOM.

A good understanding of the Photoshop DOM, and how each aspect of the DOM relates to the Photoshop application and its documents will make script writing easier.

Containment hierarchy

The Photoshop object model is a containment hierarchy, which means that objects in the model are identified partially by the objects that contain them. In Photoshop, the Application object sits at the top of the hierarchy. Applications contain a Documents collection. The Documents collection contains Document objects. A Document object contains an ArtLayers collection, a HistoryStates collection, a Layers collection, a Layersets collection, and a Channels collection. Through using commands or methods in the DOM, you can tell Photoshop documents to add and remove objects, or set or change individual object properties like color, size and shape. In the diagram below, each node in the hierarchy represents a class in the Photoshop DOM.

The Photoshop object model uses elements (AppleScript) or collections (VBScript, JavaScript) as a convenient way to group classes. We have not shown object elements or collections in the object model diagram below. Not all classes are associated with a collection. However, some key classes are grouped by elements or collection. The following elements/collections exist in Photoshop: Art Layers, Channels, Color Samplers, Count Items, Documents, Layers, Layer Comps, Layer Sets, History States, Notifiers, Path Items, Path Points Sub Path Items, and Text Fonts. See Introduction to Scripting for more information on elements and collections.

NOTE: In Photoshop, VBScript collections index from 1 rather than 0. This stands in contrast to other VBScript arrays, which index from 0.
Application and document classes

The **Application** class is the root of the Photoshop object model hierarchy. Scripts must target the appropriate application in order to run correctly. See "Targeting and Referencing the Application Object" on page 22.

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, then, save the current document, and copy and paste within the active document or between different documents. For more information on using document objects, see "Creating New Objects in a Script" on page 23 and "Using the Document object" on page 34.

Layer classes

Photoshop has two 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, defined by a **Layer Set**. 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** item is implemented as a property of the art layer. For more information on text items, see "Using the Text Item object" on page 41.
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 “Working with layer objects” on page 36.

Layer Comp class

The Layer Comp class allows you to create, manage, and view multiple versions of a layout within a single document.

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 commands or 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 “Working with Channel objects” on page 46.

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 “Working with Selection objects” on page 42.

History State class

The History State 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 “Using history state objects” on page 47.

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 “Using the Document Info object” on page 47.
Path Item, Sub Path Item, and Path Point classes

The Path Item class represents information about a drawing object, such as the outline of a shape, or a curved line. The Sub Path Item class is contained in the Path Item class, and provides the actual geometry of the shape. The Path Point Class contains information about each point in a sub path. See “Using the PathItem object” on page 49.

Preferences class

The Preferences class allows you to access and set the user preference settings. See “Working with document preferences” on page 61.

Notifier class

The Notifier object ties an event to a script. For example, if you would like Photoshop to automatically create a new document when you open the application, you could tie a script that creates a Document object to an Open Application event. For more information on notifiers, see “Using Notifier objects” on page 48.

Count Item class

The Count Item object provides scripting support for the Count Tool.

Color Sampler class

The Color Sampler object provides scripting support for the Color Sampler Tool.

Measurement Scale class

The Measurement Scale object provides scripting support for the new Measurement Scale feature that allows you to set a scale for your document.

The containment hierarchy and the Photoshop user interface

The following table provides describes how each object relates to the Photoshop user interface.

<table>
<thead>
<tr>
<th>Object name</th>
<th>Description</th>
<th>To create this object without using a script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>The Photoshop application.</td>
<td>Start the Photoshop application.</td>
</tr>
<tr>
<td>Document</td>
<td>The working object, in which you create layers, channels, actions, and so on. In a script, you name, open, or save a document as you would a file in the application.</td>
<td>In Photoshop, choose File &gt; New or File &gt; Open.</td>
</tr>
<tr>
<td>Selection</td>
<td>The selected area of a layer or document.</td>
<td>Choose the marquee or lasso tools and drag your mouse.</td>
</tr>
<tr>
<td>Object name</td>
<td>Description</td>
<td>To create this object without using a script</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Path Item</td>
<td>A drawing object, such as the outline of a shape or a straight or curved line</td>
<td>Choose the path selection or pen tools and draw a path with the mouse.</td>
</tr>
<tr>
<td>Channel</td>
<td>Pixel information about an image's color.</td>
<td>Choose <strong>Window &gt; Channels</strong>.</td>
</tr>
<tr>
<td>Art Layer</td>
<td>A layer class within a document that allows you to work on one element of an image without affecting other elements in the image.</td>
<td>Choose <strong>Layer &gt; New &gt; Layer</strong> or <strong>Window &gt; Layers</strong>.</td>
</tr>
<tr>
<td>Layer Set</td>
<td>A collection of Art Layer objects. Group is the current name in the Photoshop UI. Layer Set was the name used in an earlier version of Photoshop. The Object name stays the same to keep backward compatibility.</td>
<td>Choose <strong>Layer &gt; New &gt; Group</strong>.</td>
</tr>
<tr>
<td>Layer Comp</td>
<td>A snapshot of a state of the layers in a document.</td>
<td>Choose <strong>Window &gt; Layer Comp</strong>. Then select the New Layer Comp icon.</td>
</tr>
<tr>
<td>Document Info</td>
<td>Metadata about a Document object.</td>
<td>Choose <strong>File &gt; File Info</strong>.</td>
</tr>
<tr>
<td><strong>NOTE:</strong> Metadata is any data that helps to describe the content or characteristics of a file, such as file name, creation date and time, author name, the name of the image stored in the file, etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notifier</td>
<td>Notifies a script when an event occurs; the event then triggers the script to execute. For example, when a user clicks an OK button, the notifier object tells the script what to do next.</td>
<td>Choose <strong>File &gt; Scripts &gt; Script Events Manager</strong>.</td>
</tr>
<tr>
<td>Preferences</td>
<td>The application preference settings.</td>
<td>Choose <strong>Edit &gt; Preferences</strong> in Windows, or <strong>Photoshop &gt; Preferences</strong> in Mac OS.</td>
</tr>
<tr>
<td>History State</td>
<td>Stores a version of the document in the state the document was in each time you saved it.</td>
<td>Choose <strong>Window &gt; History</strong>, and then choose a history state from the History palette.</td>
</tr>
<tr>
<td><strong>NOTE:</strong> You can use a History State object to fill a Selection object or to reset the document to a previous state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color Sampler</td>
<td>Represents a color sampler in your document.</td>
<td>Choose the Color Sampler Tool, and click in the document.</td>
</tr>
<tr>
<td>Count Item</td>
<td>Represents a counted item in the document.</td>
<td>Choose the Count Tool and click in the document.</td>
</tr>
<tr>
<td>Measurement Scale</td>
<td>Represents the measurement scale for your document.</td>
<td>The Measurement Scale object cannot be created, but you can change its properties by using <strong>Analysis &gt; Measurement Scale &gt; Custom</strong>.</td>
</tr>
</tbody>
</table>
Additional objects

The Photoshop object model includes additional objects beyond the ones described in the containment hierarchy above. Most of these classes are used as types for properties or to provide information (as arguments) for commands or methods. For example:

- The `color` value (SolidColor/SolidColor) class provides the type for the background color (backgroundColor/backgroundColor) and foreground color (ForegroundColor/foregroundColor) properties of the `Application` object. See "Working with color objects" on page 51.

- Open and save options for documents are defined as classes, and these are passed to the commands that open and save documents; e.g., the `BMP save options` (BMPSaveOptions/BMPSaveOptions) class can be passed as an argument to the `save` (saveAs/saveAs) command or method. See “Opening a Document” on page 28 and “Saving a Document” on page 31.

Constants

An additional important component of the Photoshop object model for JavaScript and VBScript are constants. Constants are a type of value that defines a property. For example, with the `kind` property of an `ArtLayer` object, you can define only specific kinds that Photoshop allows. For general information about constants, see *Introduction to Scripting*.

**NOTE:** Throughout this document, actual values of enumerations for VBScript are given using the following format:

```
newLayerRef.Kind = 2 '2 indicates psLayerKind --> 2 (psTextLayer)
```

The `'` before the explanation creates a comment and prevents the text to the right of the `'` from being read by the scripting engine. For more information about using comments, see *Introduction to Scripting*.

For example, look up the art `ArtLayer` object in either the Adobe Photoshop CS6 JavaScript Scripting Reference or in the Adobe Photoshop CS6 Visual Basic Scripting Reference. One of the properties of this object is `Kind(kind)`. The value type for that property contains a link to the constant that define the allowed values for the property. For VBScript, the constant is `PSLayerKind`, for JavaScript, the constant is `LayerKind`. Click the link to view the values you can use to define the `kind` property.

**NOTE:** Different objects can use the same property name with different constant values. The constant values for the `Channel` object’s `kind` property are different than the constant values for the `ArtLayer` object’s `kind` property.
Creating a sample Hello World script

This section demonstrates a very simple script in each of the three scripting languages for Photoshop. Traditionally, the first thing to accomplish in any programming environment is the display of a "Hello World" message.

Our Hello World scripts will do the following:

1. Open the Photoshop application.
2. Create a new Document object.
   
   When we create the document, we will also create a variable named docRef and then assign a reference to the document as the value of docRef. The document will be 4 inches wide and 2 inches high.

3. Create an Art Layer object.
   
   In our script, we will create a variable named artLayerRef and then assign a reference to the Art Layer object as the value of artLayerRef.

4. Define artLayerRef as a text item.

5. Set the contents of the text item to "Hello World".

NOTE: We will also include comments throughout the scripts. In fact, because this is our first script, we will use comments to excess.

These steps mirror a specific path in the containment hierarchy, as illustrated below.
Creating and running an AppleScript

You must open the Apple® Script Editor application in order to complete this procedure.

**NOTE:** The default location for the Script Editor is Applications > AppleScript > Script Editor.

To create and run your first Photoshop AppleScript:

1. Enter the following script in the Script Editor:

   **NOTE:** The lines preceded by “--” are comments. Entering the comments is optional.

   ```
   -- Sample script to create a new text item and
   -- change its contents.
   --target Photoshop CS6
   tell application "Adobe Photoshop CS6"
   
   -- Create a new document and art layer.
   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

   -- 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
   ```

2. Click **Run** to run the script. Photoshop creates a new document, adds a new layer, changes the layer’s type to text and sets the text to “Hello, World”

   **NOTE:** If you encounter errors, refer to *Introduction to Scripting*, which has a section on AppleScript debugging.
Creating and running a VBScript

Follow these steps to create and run a VBScript that displays the text *Hello World!* in a Photoshop document.

**To create and run your first Photoshop VBScript:**

1. Type the following script into a script or text editor.

   **NOTE:** Entering comments is optional.

   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 2x4 inch document and assign it to a variable.
   Dim docRef
   Dim artLayerRef
   Dim textItemRef
   Set docRef = appRef.Documents.Add(2, 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

2. Save file as a text file with a `.vbs` file name extension.

3. Double-click the file in Windows Explorer to run the script.

   The script opens Photoshop.

Creating and running a JavaScript

Follow these steps to create and run a JavaScript that displays the text *Hello World!* in a Photoshop document.

Because you will be actually using Photoshop to run your JavaScripts, it is not necessary to include code that opens Photoshop at the beginning of the script.

**NOTE:** Adobe has created the Extend Script scripting language to augment JavaScript for use with Photoshop. You can use the Extend Script command `target` to target the Photoshop application and create the ability to open JavaScripts that manipulate Photoshop from anywhere in your file system. See the “Script UI” chapter of the *JavaScript Tools Guide* for more information.
To create and run your first Photoshop JavaScript:

1. Type the following script.

   **NOTE:** Entering comments is optional.

   ```javascript
   // 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 2x4 inch document and assign it to a variable
   var docRef = app.documents.add( 2, 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
   ```

2. Save file as a text file with a `.jsx` file name extension in the Presets/Scripts folder in your Adobe Photoshop CS6 directory.

   **NOTE:** You must place your JavaScripts in the Presets/Scripts folder in order to make the scripts accessible from the **File > Scripts** menu in Photoshop. The scripts do not appear on the **File > Scripts** menu until you restart the application.

   **NOTE:** Photoshop also supports JavaScript files that use a `.js` extension.

3. Do either of the following:

   ➢ If Photoshop is already open, choose **File > Scripts > Browse**, and then navigate to the Presets > Scripts folder and choose your script.

   ➢ Start or restart Photoshop, and then choose **File > Scripts**, and then select your script from the **Scripts** menu.
This chapter demonstrates several techniques for using the Photoshop Document Object Model (DOM) to create scripts to use specifically with Photoshop.

You will also learn how to use the reference manuals and object model browsers to find information about the objects, classes, properties, commands, and even some values (called constants or enumerations) you can use to create AppleScripts, VBScript scripts, and JavaScripts for Photoshop.

**Tip:** Throughout this chapter, the explanation of how to create a script is followed by instructions for locating information about the specific elements used in the script. Using these instructions will help you quickly understand how to script Photoshop.

### Viewing Photoshop Objects, Commands, and Methods

The Photoshop reference material for each of the three scripting languages is found in the reference manuals provided in this installation:

- Adobe Photoshop CS6 AppleScript Scripting Reference
- Adobe Photoshop CS6 Visual Basic Scripting Reference
- Adobe Photoshop CS6 JavaScript Scripting Reference

In addition, you can also access reference material by using the associated object model browser for each language:

- For AppleScript, use the AppleScript Script Editor to view the Photoshop AppleScript Dictionary.
- For VBScript, use the VBA editor in Microsoft Word, or the Visual Basic Object Browser in Visual Basic, or Visual Studio.
- For JavaScript, use the ExtendScript Object Model Viewer. See the JavaScript Tools Guide for more information.

### Viewing Photoshop’s AppleScript dictionary

You use Apple’s Script Editor application to view the dictionary.

**Note:** The default location for the Script Editor is Applications > AppleScript > Script Editor.

**To view the AppleScript dictionary:**

1. In Script Editor, choose **File > Open Dictionary**.
   
   Script Editor displays an Open Dictionary dialog.

2. Choose Adobe Photoshop CS6, and then click **Open**.
   
   Script Editor opens Photoshop and then displays the Photoshop dictionary, which lists objects as well as the commands, properties and elements associated with each object. The dictionary also lists the parameters for each command.
Targeting and Referencing the Application Object

Because you run your AppleScript and VBScript scripts from outside the Photoshop application, the first thing your script should do is indicate that the commands be executed in Photoshop.

**NOTE:** In JavaScript, you do not need to target the `Application` object because you open the scripts from the Photoshop application itself. (See “Creating and running a JavaScript” on page 19.)
AS  To target Photoshop in AppleScript, you must enclosing your script in the following statements:

```applescript
tell application "Adobe Photoshop CS6"
  ...  
end tell
```

**NOTE:** Because you include all commands in the `tell` block, there is no need to reference the `Application` object throughout the script.

VBS  In VBScript, do the following to target the application:

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

JS  In JavaScript, because you do not need to reference an `Application` object, all properties and methods of the application are accessible without any qualification. You can reference the application as part of the containment hierarchy or leave it out, whichever makes your scripts easier for you to read.

To reference the `Application` object, use the pre-defined global object `app`, rather than the class name.

The following statements are equivalent:

```javascript
var docRef = app.documents[1]
```

```javascript
var docRef=documents[1]
```

**NOTE:** Many JavaScript samples throughout this guide do not reference the `Application` object.

Creating New Objects in a Script

To create a new document in the Photoshop application, you select **File > New**. To create other types of objects within a document, such as a layer, channel, or path, you use the Window menu or choose the **New** icon on the appropriate palette. This section demonstrates how to accomplish these same tasks in a script.

To create an object in a script, you name the type of object you want to create and then use the following command:

- **AS:** `make`
- **VBS:** `Add`
- **JS:** `add()`

As you can see in the "Photoshop Object Model" on page 11, the `Document` object contains all other objects except the `Application`, `Notifier`, and `Preferences` objects. Therefore, you must reference the `Document` object when adding objects other than `Document` and `Notifier` objects to your script. (It is not possible to add a new `Preferences` object.)

**NOTE:** In VBScript and JavaScript, you use the object's collection name to name the object type. For example, you add a document to the `Documents` collection; you add an art layer to the `art layers` collection. See *Introduction to Scripting* for more information on elements and collections.
The following statement creates a `Document` object in an AppleScript.

```
make new document
```

You can also use the `set` command to create a variable to hold a reference to a new document. In the following example, the variable named `docRef` holds a reference to the new document:

```
set docRef to make new document
```

To create an object other than a document, you must reference the `Document` object that contains the object. The following sample creates an art layer in the document contained in the variable named `docRef`.

```
make new art layer in docRef
```

**NOTE:** When you create an object in AppleScript, you actually add the object to an element the same way you add a VBS or JavaScript object to a collection. However, in AppleScript, the element name is implied in the `make` or `set` statement. For example, the statement:

```
make new document
```

actually means:

```
make new document in the documents element
```

Do the following to find out more about creating objects in an AppleScript:


- To find out which commands can be used with an object, look up the object in the *Adobe Photoshop CS6 AppleScript Scripting Reference*. If an object has valid commands, there will be a “Valid Commands” list at the end of the object description.

In VBS, you can use the Add method only with the collection name. The Add method is not valid with objects other than collection objects. Also, in VBS, you must reference the `Application` object when creating or referring to an object in your script.

For example, to create a document in a VBS script, you **cannot** use the object name, as in the following sample, which creates a `Document` object:

```
appRef.Document.Add()
```

You must use the collection name, which is a plural form of the object name, as follows:

```
appRef.Documents.Add()
```

**NOTE:** In this sample statement, the `Application` object is referenced via a variable named `appRef`. See “Targeting and Referencing the Application Object” on page 22 for more information.

To add an `ArtLayer` object, you must reference both the `Application` and `Document` objects that will contain the art layer. The following sample references the `Application` object using the variable `appRef` and the `Document` object using the document’s index rather than the documents name.

```
appRef.Documents(1).ArtLayers.Add()
```

**NOTE:** In Photoshop, VBS collections index from 1 rather than 0. That is to say, the first document created has index 1, rather than index 0.
If you look up in the Document object in the Adobe Photoshop CS6 Visual Basic Scripting Reference or in the Visual Basic Object Browser, you will see that there is no Add() method for the object. However, the Add() method is available for the Documents object. Similarly, the ArtLayer object does not have an Add() method; the ArtLayers object does.

**NOTE:** The Layers object is an exception because, although it is a collection object, it does not include an Add() method. The Layers collection includes both ArtLayer and LayerSet objects, which are created with the Add method on either the ArtLayers or LayerSets collections. For more information, look up the Layers object in the Adobe Photoshop CS6 Visual Basic Scripting Reference.

**JS**

In JavaScript, you can use the add() method only with the collection name. The add() method is not valid with objects other than collection objects.

Similar to VBScript, the JavaScript statement to create a document is:

```javascript
documents.add()
```

and not:

```javascript
document.add()
```

**NOTE:** You can include an Application object reference if you wish. The following statement is equivalent to the previous sample:

```javascript
app.documents.add()
```

To add an ArtLayer object, you must reference the Document object that contains the layer, and use the add() method for the ArtLayers collection, using the artLayers property of Document.

```javascript
documents[0].artLayers.add()
```

As with VBScript, the add() method is associated with the JavaScript Documents object but not with the Document object. Similarly, the ArtLayer object does not have an add() method; the ArtLayers object does.

**NOTE:** The Layers collection object does not include an add() method. For more information, look up the Layers object in the Adobe Photoshop CS6 JavaScript Scripting Reference.

### Setting the Active Object

To work on an object in the Photoshop application, you must make the object the front-most, or active object. For example, to work in a layer, you must first bring the layer to the front.

In scripting, the same rule applies. If your script creates two or more documents, the commands and methods in your script are executed on the active document. Therefore, to ensure that your commands are acting on the correct document, it is good programming practice to designate the active document before executing any commands or methods in the script.

To set an active object, do the following:

- In AppleScript, you use the current property of the parent object.
- In VBScript, you use the ActiveObject property of the parent object (such as ActiveDocument or ActiveLayer).
- In JavaScript, you use the activeObject property of the parent object (such as activeDocument or activeLayer).
NOTE: The parent object is the object that contains the specified object. For example, the application is the parent of the document; a document is the parent of a layer, selection, or channel.

For example, if you look at the Application object in the Adobe Photoshop CS6 JavaScript Scripting Reference, or in the ExtendScript Object Model Viewer, you find one of its properties is `activeDocument`; if you look at the Document object, you will find `activeLayer` and `activeHistoryState` as properties. Similarly, if you look at application in the Adobe Photoshop CS6 AppleScript Scripting Reference, or in the Photoshop AppleScript Dictionary, you find it has the property of `current`, and so on.

For sample scripts that set active objects, see the following sections.

➤  “Setting the active document” on page 26
➤  “Setting the active layer” on page 27
➤  “Setting the active channels” on page 28

## Setting the active document

The following examples demonstrate how to set the active document.

**AS**

```as
--create 2 documents
set docRef to make new document with properties ¬
    {width:4 as inches, height:4 as inches}
set otherDocRef to make new document with properties ¬
    {width:4 as inches, height:6 as inches}

--make docRef the active document
set current document to docRef
--here you would include command statements
--that perform actions on the active document. Then, you could
--make a different document the active document

--use the current document property of the application class to
--bring otherDocRef front-most as the new active document
set current document to otherDocRef
```

**VBS**

```vbs
'Create 2 documents
Set docRef = app.Documents.Add ( 4, 4)
Set otherDocRef = app.Documents.Add (4,6)

'make docRef the active document
Set app.ActiveDocument = docRef
'here you would include command statements
'make a different document the active document

'use the ActiveDocument property of the Application object to
'bring otherDocRef front-most as the new active document
Set app.ActiveDocument = otherDocRef
```
JS

// Create 2 documents
var docRef = app.documents.add(4, 4)
var otherDocRef = app.documents.add(4, 6)

// make docRef the active document
app.activeDocument = docRef

// here you would include command statements
// that perform actions on the active document. Then, you could
// make a different document the active document

// use the activeDocument property of the Application object to
// bring otherDocRef front-most as the new active document
app.activeDocument = otherDocRef

Setting the active layer

The following examples demonstrate how to use the current layer (ActiveLayer/activeLayer) property of the Document object to set the active layer. In order to set the active layer for a document, the document itself must be the current document.

AS

set current layer of current document to layer “Layer 1” of current document

NOTE: By default, Photoshop names the layers “Layer 1”, “Layer 2”, etc.

VBS

' This example assumes appRef and docRef have been previously defined and assigned
' to the application object and a document object that contains at least one layer.
appRef.ActiveDocument = docRef
docRef.ActiveLayer = docRef.Layers(1)

Look up the ActiveLayer property on the Document object in the Adobe Photoshop CS6 Visual Basic Scripting Reference, or in the Visual Basic Object Browser.

NOTE: You can also use the name of the layer to indicate which layer to use. By default, Photoshop names the layers “Layer 1”, “Layer 2”. See “Referencing ArtLayer objects” on page 38.

JS

// This example assumes docRef has been previously defined and assigned to a
// document object that contains at least one layer.
activeDocument = docRef
docRef.activeLayer = docRef.layers[0]


NOTE: You can also use the name of the layer to indicate which layer to use. By default, Photoshop names the layers “Layer 1”, “Layer 2”. See “Referencing ArtLayer objects” on page 38.
CHAPTER 3: Scripting Photoshop

Setting the active channels

More than one channel can be active at a time, so the current channels (ActiveChannels/activeChannels) property of the Document object takes an array of channels as a value. In order to set the active channels of a document, it must be the active document.

AS

Set the active channels to the first and third channel using a channel array:

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

Alternatively, select all component channels using the component channels property of the Document object.

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

VBS

Set the active channels of the active document to the first and third channel using a channel array:

' This example assumes docRef is already the ActiveDocument
Dim theChannels
theChannels = Array(docRef.Channels(1), docRef.Channels(3))
docRef.ActiveChannels = theChannels

Alternatively, select all component channels using the ComponentChannels property of the Document object:

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

JS

Set the active channels to the first and third channel using a channel array:

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

Alternatively, select all component channels by using the componentChannels property of the Document object:

app.activeDocument.activeChannels =
  activeDocument.componentChannels

Opening a Document

You use the open/Open/open() command of the Application object to open an existing document. You must specify the document name (that is, the path to the file that contains the document) with the command.

Opening a file with default file format

Because Photoshop supports many different file formats, the open/Open/open() command lets you specify the format of the document you are opening. If you do not specify the format, Photoshop infers the type of file for you, which is called the file’s default format. The following examples open a document by inferring the most appropriate format to use:
CHAPTER 3: Scripting Photoshop

Opening a Document

AS

set theFile to alias "Applications:Documents:MyFile"
open theFile

VBS

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

JS

var fileRef = File(app.path + "/Samples/Fish.psd")
var docRef = app.open(fileRef)

Notice that in JavaScript, you must create a File object and then pass a reference to the object to the open() command.

Specifying file formats to open

For the document types on the following list, you can set options to specify how the document will be opened, such as the height and width of the window in which the document is opened, which page to open to in a multi-page file, etc.

- PhotoCD
- CameraRaw
- RawFormat
- Adobe PDF
- EPS

To find out which options you can set for each of file type, look up the properties for the OpenOptions objects that begin with the file format name. For example:

- In the Adobe Photoshop CS6 AppleScript Scripting Reference look up the Photo CD open options class or the EPS open objects class.

- In the Adobe Photoshop CS6 Visual Basic Scripting Reference, or the Adobe Photoshop CS6 JavaScript Scripting Reference, look up the PhotoCDOpenOptions or EPSOpenOptions objects.

The following examples demonstrate how to open a generic (multi-page/multi-image) PDF document with the following specifications:

- The document will open in RGB mode with a resolution of 72 pixels/inch.
- Antialiasing will be used to minimize the jagged appearance of the edges of images in the document.
CHAPTER 3: Scripting Photoshop

Opening a Document

➤ The document will open to page 3.
➤ The document's original shape will change to conform to the height and width properties if the original shape is not twice as wide as it is tall.

AS

tell application "Adobe Photoshop CS6"
    set myFilePath to alias "OS X 10.5.8 US:Users:psauto:Desktop:opal_screen.pdf"
    with timeout of 300 seconds
        open myFilePath as PDF with options ¬
            {class:PDF open options, ¬
                mode:RGB, resolution:72, use antialias:true, page:3}
    end timeout
end tell

VBS

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

'Remember unit settings and set to values expected by this script
Dim originalRulerUnits
originalRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = 1 'value of 1 = psPixels

'Create a PDF option object
Dim pdfOpenOptionsRef
Set pdfOpenOptionsRef = CreateObject("Photoshop.PDFOpenOptions")
pdfOpenOptionsRef.AntiAlias = True
pdfOpenOptionsRef.Mode = 2 'psOpenRGB
pdfOpenOptionsRef.Resolution = 72
pdfOpenOptionsRef.Page = 3

' open the file
Dim docRef
Set docRef = appRef.Open("C:\PDFFiles\MyFile.pdf", pdfOpenOptionsRef)

'Restore unit setting
appRef.Preferences.RulerUnits = originalRulerUnits

JS

NOTE: The ExtendScript File object expects Universal Resource Identifier (URI) notation. Please see the JavaScript Tools Guide for more information.

// 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("/c/pdffiles/myfile.pdf")

// Create a PDF option object
var pdfOpenOptions = new PDFOpenOptions
pdfOpenOptions.antiAlias = true
pdfOpenOptions.mode = OpenDocumentMode.RGB
pdfOpenOptions.resolution = 72
pdfOpenOptions.page = 3
// open the file
app.open( fileRef, pdfOpenOptions )

// restore unit settings
app.preferences.rulerUnits = originalRulerUnits
Saving a Document

Options for saving documents in Photoshop are illustrated below. To find out which properties you can specify for a specific file format save option, look up the object that begins with the file format name. For example, to find out about properties for saving an `.eps` file, do the following:

➤ In the *Adobe Photoshop CS6 AppleScript Scripting Reference*, look up the class `EPS` save options.

➤ In the *Adobe Photoshop CS6 Visual Basic Scripting Reference* or in the *Adobe Photoshop CS6 JavaScript Scripting Reference* look up `EPSSaveOptions`.

![Save Classes Diagram]

**NOTE:** It is important to note that the `Open` and `Save` formats are not identical. See “Specifying file formats to open” on page 29 for comparison.

**NOTE:** The following optional formats are available only when installed explicitly:

➤ Alias PIX

➤ Electric Image

➤ SGI RGB

➤ Wavefront RLA

➤ SoftImage

The following scripts save a document as a `.jpeg` file.

```as
tell application "Adobe Photoshop CS6"
make new document
set myFile to "OS X 10.5.8 US:Users:psauto:Desktop:Rat.jpg"
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
```
CHAPTER 3: Scripting Photoshop

Setting Application Preferences

Your script can set application preferences such as color picker, file saving options, guide-grid-slice settings, and so on.

**NOTE:** The properties in the settings class/Preferences object correlate to the Photoshop CS6 Preferences dialog options, which you display by choosing Photoshop > Preferences on Mac OS or Edit > Preferences in Windows versions of Photoshop. For explanations of individual preferences, please refer to Photoshop Help.

**AS**
You use properties of the settings class to set application preferences in AppleScript. The following script sets ruler and type unit settings:

```applescript
set ruler units of settings to inch units
type units of settings to pixel units
```

In the *Adobe Photoshop CS6 AppleScript Scripting Reference*, or in the Photoshop AppleScript Dictionary, look up class settings-object to view all of the settings properties you can use.

**VBS**
The Preferences object is a property of the Application object. When you use the Preferences object in a VBScript script, you must indicate its containment in the Application object.

```vbscript
appRef.Preferences.RulerUnits = 2 'for PsUnits --> 2 (psInches)
appRef.Preferences.TypeUnits = 1 'for PsTypeUnits --> 1 (psPixels)
```

In the *Adobe Photoshop CS6 Visual Basic Scripting Reference*, or in the Visual Basic Object Browser, look up the Preferences object to view all of the settings properties you can use. Additionally, look up the Preferences property on the Application object.

**VBS**
```vbscript
Dim appRef, docRef
Set appRef = CreateObject("Photoshop.Application")
Set docRef = appRef.Documents.Add()

Set jpgSaveOptions = CreateObject("Photoshop.JPEGSaveOptions")
jpgSaveOptions.EmbedColorProfile = True
jpgSaveOptions.FormatOptions = 1 'for psStandardBaseline
jpgSaveOptions.Matte = 1 'for psNoMatte
jpgSaveOptions.Quality = 1
appRef.ActiveDocument.SaveAs "c:\temp\myFile2", _
    jpgSaveOptions, True, 2 'for psLowercase
```

**JS**
```javascript
app.documents.add( 4, 4 )
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)
```
CHAPTER 3: Scripting Photoshop

Allowing or Preventing Dialogs

It is important to be able to control dialogs properly from a script. If a dialog appears, 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.

You use the `display dialogs` (`DisplayDialogs/displayDialogs`) property of the `Application` object to control whether or not dialogs are displayed.

**NOTE:** Using dialogs in your script is roughly equivalent to using stops in a Photoshop action.

The following script prevents dialogs from being displayed:

```javascript
set display dialogs to never
```

In the Adobe Photoshop CS6 AppleScript Scripting Reference or in the Photoshop AppleScript Dictionary, look up the Class `application` to find the values you can use for the `display dialogs` property.

To set dialog preferences, you use the `DisplayDialogs` property of the `Application` object.

```vbs
appRef.DisplayDialogs = 3
'for PsDialogModes --> 3 (psDisplayNoDialogs)
```

Note that, because `DisplayDialogs` is a property of the `Application` object, you must reference the `Application` object in the script to get to the property.

To set dialog preferences, you use the `displayDialogs` property of the `Application` object.

```javascript
displayDialogs = DialogModes.NO
```

In the Adobe Photoshop CS6 JavaScript Scripting Reference, or in the ExtendScript Object Model Viewer, look up the `Application` object property `displayDialogs`, and then look up the constant `DialogModes`.

Working with the Photoshop Object Model

This section contains information about using the objects in the Photoshop Object Model. For information on object models, see *Introduction to Scripting* and “Photoshop Object Model” on page 11.
Using the Application object

This section describes how and when to use the Application object in a script. It also describes how to use some properties of the Application object.

You use the properties and commands of the Application object to work with Photoshop functionality and objects such as the following:

➤ **Global Photoshop settings or preferences, such as unit values or color settings** — See “Setting Application Preferences” on page 32.

➤ **Documents** — You can add or open documents and set the active document. “Opening a Document” on page 28 and “Setting the Active Object” on page 25.

➤ **Actions** — You can execute actions created either via scripting or using the Actions palette in the Photoshop application. See “Action Manager” on page 73.

You can use Application object properties to get information such as the following:

➤ A list of fonts installed on the system.
  - AS: set theFonts to fonts
    - **NOTE:** In AppleScript, fonts is a separate collection, and does not require a reference to the application object to retrieve it.
  - VBS: Set fontsInstalled = AppRef.Fonts
  - JS: var fontsInstalled = app.fonts

➤ The amount of unused memory available to Adobe Photoshop, using the free memory (FreeMemory/freeMemory) property of the Application object.

➤ The location of the Preferences folder, using the preferences folder (PreferencesFolder/preferencesFolder) property of the Application object.

For further information, look up the properties of the Application object in the reference manual or the object browser of the language you are using.

Using the Document object

The Document object can represent any open document in Photoshop. You can think of a Document object as a file; you can also think of it as a canvas. You work with the Document object to do the following:

➤ **Access script objects contained in the Document object, such as ArtLayer or Channel objects.** See “Photoshop Object Model” on page 11 for more information.

➤ **Manipulate a specific Document object**, using commands or methods. For example, you could crop, rotate or flip the canvas, resize the image or canvas, and trim the image. See “Manipulating a document object” on page 35 for a demonstration.

➤ **Get the active layer.** See “Setting the active layer” on page 27.

➤ **Save the current document.** See “Saving a Document” on page 31.

➤ **Copy and paste within the active document or between different documents.** See “Understanding Clipboard Interaction” on page 54.
Manipulating a document object

The following examples demonstrate how to do the following:

➤ Save the existing ruler unit preferences, and set ruler units to inches
➤ Open an existing file as a document (using file Ducky.tif)
➤ Change the size of the image to 4 inches wide and 4 inches high.
➤ Change the size of the document window (or canvas) to 4 inches high and 4 inches wide.
➤ Trim the top and bottom of the image.
➤ Crop the image.
➤ Flip the entire window.
➤ Restore the original ruler units.

NOTE: See "Setting Application Preferences" on page 32 for information on ruler units.

AS
tell application "Adobe Photoshop CS6"
    set saveUnit to ruler units of settings
    set ruler units of settings to inch units
    set duckFile to alias ¬
        "OS X 10.5.8 US:Applications:Adobe Photoshop CS6:Samples:Ducky.tif"
    open duckFile
    set docRef to current document
    resize image docRef width 4 height 4
    resize canvas docRef width 4 height 4
    trim docRef basing trim on top left pixel with top trim ¬
        and bottom trim without left trim and right trim
    set ruler units of settings to pixel units
    crop current document bounds {100, 200, 400, 500} angle 45 width 20 height 20
    flip canvas docRef direction horizontal
    set ruler units of settings to saveUnit
end tell

VBS
Dim appRef, docRef
Set appRef = CreateObject("Photoshop.Application")

'save original ruler units, then set ruler units to inches
startRulerUnits = appRef.Preferences.RulerUnits
appRef.Preferences.RulerUnits = 2 'for PsUnits --> 2 (psInches)
Set docRef = appRef.Open(appRef.Path & "\Samples\Ducky.tif")
docRef.ResizeImage 4,4
docRef.ResizeCanvas 4,4

'Trim the document with
' type = 1 (psTopLeftPixel)
' top=true, left=false, bottom=true, right=false
docRef.Trim 1,True,False,True,False

'the crop command uses unit values
'so change the ruler units to pixels
appRef.Preferences.RulerUnits = 1 ' (psPixels)
'Crop the document with
  angle=45, width=20,height=20
docRef.Crop Array(100,200,400,500),45,20,20
docRef.FlipCanvas 1 'psHorizontal

'restore ruler units
appRef.Preferences.RulerUnits = startRulerUnits

JS

//save original ruler units, then assign it to inches
startRulerUnits = app.preferences.rulerUnits
app.preferences.rulerUnits = Units.INCHES

//get a reference to the file, and open it
var fileRef = new File(app.path + "\samples/duddy.tif")
var docRef = app.open(fileRef)

//this sample script assumes the ruler units have been set to inches
docRef.resizeImage( 4,4 )
docRef.resizeCanvas( 4,4 )
docRef.trim(TrimType.TOPLEFT, true, false, true, false)

//the crop command uses unit values
//so change the ruler units to pixels
app.preferences.rulerUnits = Units.PIXELS
docRef.crop (new Array(100,200,400,500), 45, 20, 20)
docRef.flipCanvas( Direction.HORIZONTAL )

//restore original preferences
app.preferences.rulerUnits = startRulerUnits

Working with layer objects

The Photoshop object model contains two types of layer objects:

➤ **ArtLayer** objects, which can contain image contents and are basically equivalent to Layers in the Photoshop application.

  **NOTE:** An ArtLayer object can also contain text if you use the *kind* property to set the ArtLayer object’s type to text layer.

➤ **Layer Set** objects, which can contain zero or more ArtLayer objects.

When you create a layer you must specify whether you are creating an **ArtLayer** or a **Layer Set**.

**NOTE:** Both the ArtLayer and LayerSet objects have corresponding collection objects, ArtLayers and LayerSets, which have an *add/Add/add()* command. You can reference, but not add, ArtLayer and LayerSet objects using the Layers collection object, because, unlike other collection objects, it does not have an *add/Add/add()* command.
Creating an ArtLayer object

The following examples demonstrate how to create an ArtLayer object filled with red at the beginning of the current document.

**AS**
```
tell application "Adobe Photoshop CS6"
    make new document
    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
```

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

' Create a new art layer at the beginning of the current document
Dim docRef
Dim layerObj
Set docRef = appRef/Documents/Add()
Set layerObj = appRef/ActiveDocument/ArtLayers/Add
layerObj.Name = "MyBlendLayer"
layers[0].BlendMode = 2 '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
Set colorObj = CreateObject("Photoshop.SolidColor")
colorObj.RGB.Red = 255
colorObj.RGB.Green = 0
colorObj.RGB.Blue = 0

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

**JS**
```
// make a new document
app.documents.add()

// 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)
```
Creating a Layer Set object

The following examples show how to create a Layer Set object after the creating the first ArtLayer object in the current document:

**AS**
```
tell application "Adobe Photoshop CS6"
  make new document with properties {name:"My Document"}
  make new art layer at beginning of current document
  make new layer set after layer 1 of current document
end tell
```

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

' Make a new document and a first layer in the document
appRef.Documents.Add()
appRef.ActiveDocument.ArtLayers.Add()

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

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

' Move the new layer to after the first layer
newLayerSetRef.Move layerRef, 4 'psPlaceAfter
```

**JS**
```
// make a new document and a layer in the document
app.documents.add()
app.activeDocument.artLayers.add()

// 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)
```

**Referencing ArtLayer objects**

When you create a layer in the Photoshop application (rather than a script), the layer is added to the Layers palette and given a number. These numbers act as layer names and do not correspond to the index numbers of ArtLayer objects you create in a script.

Your script—VBScript or JavaScript—will always consider the layer at the top of the list in the Layers palette as the first layer in the index. For example, if your document has four layers, the Photoshop application names them Background Layer, Layer 1, Layer 2, and Layer 3. Normally, Layer 3 would be at the top of the list in the Layers palette because you added it last.
You can use the following syntax to refer to the layers by the names given them by the Application:

**AS**  
layer 1 of layer set 1 of current document

**NOTE:** Unlike object references in JavaScript or VBScript, AppleScript object reference names do not remain constant. Refer to an AppleScript language guide or text book for information on referencing a file using either as alias or to a reference to file.

**VBS**  
Layers("Layer 3").name

**JS**  
layers["Layer 3"].name //using the collection name and square brackets for the collection

### Working with Layer Set objects

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

**VBS**  
In VBScript you can duplicate and place the layer with the same method.

Dim appRef, docRef
Set appRef = CreateObject("Photoshop.Application")

' Make a new document and a first layer in the document
Set docRef = appRef/Documents.Add()
appRef/ActiveDocument/ArtLayers.Add()

Set layerSetRef = docRef/LayerSets.Add()
Set layerRef = docRef/ArtLayers(1).Duplicate(layerSetRef, 2)

**JS**  
In JavaScript you can place the layer during the duplication method.

// create a document and an initial layer
var docRef = app.documents.add()
docRef.artLayers.add()

var layerSetRef = docRef/layerSets.add()
var layerRef = docRef/ArtLayers[0].duplicate(layerSetRef, ElementPlacement.PLACEATEND)
Linking layer objects

Scripting also supports linking and unlinking layers. You link layers together so that you can move or transform the layers in a single statement.

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

Look up the link command in the Adobe Photoshop CS6 AppleScript Scripting Reference or in the Photoshop AppleScript Dictionary.

VBS
Set layer1Ref = docRef.ArtLayers.Add()
Set layer2Ref = docRef.ArtLayers.Add()
layer1Ref.Link layer2Ref


JS
var layerRef1 = docRef.artLayers.add()
var layerRef2 = docRef.artLayers.add()
layerRef1.link(layerRef2)


Applying styles to layers

NOTE: This procedure corresponds directly to dragging a style from the Photoshop Styles palette to a layer.

Your script can apply styles to an ArtLayer object. To apply a style in a script, you use the apply layer style/ApplyStyle/applyStyle() command with the style’s name as an argument enclosed in straight double quotes.

NOTE: The layer style names are case sensitive.

Please refer to Photoshop Help for a list of styles and for more information about styles and the Styles palette.

The following examples set the Puzzle layer style to the layer named “L1.”

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

Look up the apply layer style command in the Adobe Photoshop CS6 AppleScript Scripting Reference or in the Photoshop AppleScript Dictionary.

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

Look up ApplyStyle as a method of the ArtLayer object in the Adobe Photoshop CS6 Visual Basic Scripting Reference, or in the Visual Basic Object Browser.

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

Using the Text Item object

You can change an existing `ArtLayer` object to a text layer, that is, a `Text Item` object, if the layer is empty. Conversely you can change a `Text Item` object to an `ArtLayer` object. This “reverse” procedure rasterizes the text in the layer object.

The `Text Item` object is a property of the `ArtLayer` object. However, to create a new text layer, you must create a new `ArtLayer` object and then set the art layer’s `kind/Kind/kind` property to `text layer` (2 (psTextLayer) / LayerKind.TEXT).

To set or manipulate text in a text layer, you use the `text-object (TextItem/TextItem)` object, which is contained in the `text object/TextItem/textItem` property of the `ArtLayer` object.

Creating a Text Item object

The following examples create an `ArtLayer` object and then use the `kind` property to convert it to a text layer.

**AS**

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

**VBS**

```plaintext
set newLayerRef = docRef.ArtLayers.Add()
newLayerRef.Kind = 2
'2 indicates psTextLayer
```

**JS**

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

See “Photoshop Object Model” on page 11 for information on the relationship between `ArtLayer` objects and `TextItem` objects.

Also, look up the following:

- The `kind` and `text object` properties of the class `art layer` in the *Adobe Photoshop CS6 AppleScript Scripting Reference* or in the Photoshop AppleScript Dictionary.

Determining a layer’s kind

The following examples use an `if` statement to check whether an existing layer is a text layer.

**AS**

```plaintext
if (kind of layerRef is text layer) then
...
endif
```

**VBS**

```plaintext
If layerRef.Kind = 2 Then '2 indicates psTextLayer
...
End If
```

**JS**

```plaintext
if (newLayerRef.kind == LayerKind.TEXT)
{...}
```
Adding and manipulating text in a Text Item object

The following examples add and right-justify text in a text layer.

**AS**

```plaintext
set layerRef to make new art layer in current document with properties-
   {kind:text layer}
set contents of text object of layerRef to "Hello, World!"
set justification of text object of layerRef to right
```

**VBS**

```plaintext
Set textLayerRef = docRef.ArtLayers.Add()
textLayerRef.Kind = 2
textLayerRef.Name = "my text"

Set textItemRef = docRef.ArtLayers("my text").TextItem
textItemRef.Contents = "Hello, World!"
textItemRef.Justification = 3
'3 = psRight (for the constant value psJustification)
```

**JS**

```plaintext
var textLayerRef = docRef.artLayers.add()
textLayerRef.name = "my text"
textLayerRef.kind = LayerKind.TEXT

var textItemRef = docRef.artLayers["my text"].textItem
textItemRef.contents = "Hello, World!"
textItemRef.justification = Justification.RIGHT
```

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

The text-object properties height, width, and leading are valid only when the text item’s kind property is set to paragraph text.

To familiarize yourself with this objects, properties, and commands in the scripting references, do the following:

➤ In the Adobe Photoshop CS6 AppleScript Scripting Reference or in the Photoshop AppleScript Dictionary, look up the text-object properties and methods.

➤ In the Adobe Photoshop CS6 Visual Basic Scripting Reference, or in the Visual Basic Object Browser look up the TextItem property of the ArtLayer object. To find the properties and methods you can use with a text layer, look up the TextItem object.

In the Adobe Photoshop CS6 JavaScript Scripting Reference, or in the ExtendScript Object Model Viewer, look up the textItem property of the ArtLayer object. To find the properties and methods you can use with a text layer, look up the TextItem object.

Working with Selection objects

You use a Selection object to allow your scripts to act only on a specific, selected section of your document or a layer within a document. For example, you can apply effects to a selection or copy the current selection to the clipboard.
The `Selection` object is a property of the `Document` object. Look up the following for more information:

- In the *Adobe Photoshop CS6 AppleScript Scripting Reference* or in the Photoshop AppleScript Dictionary, look up the command `select`. Also, look up the `selection` property of the `Document` object, and the `selection-object`.

- In the *Adobe Photoshop CS6 Visual Basic Scripting Reference*, or in the Visual Basic Object Browser, look up `Selection` as a property of the `Document` object. Also, look up the `Select` as a method of the `Selection` object.

- In the *Adobe Photoshop CS6 JavaScript Scripting Reference*, or in the ExtendScript Object Model Viewer, look up `selection` as a property of the `Document` object. Also, look up the `select` as a method of the `Selection` object.

**NOTE:** You cannot create a new `Selection` object. The property `selection` (Selection/selection) on the `Document` object contains a pre-existing selection object for the document. Use the `select` (Select/select) command to specify the area for the selection.

### Creating and defining a selection

To create a selection, you use the `select/Select/select()` command of the `Selection` object.

You define a selection by specifying the coordinates on the screen that describe the selection's corners. Since your document is a 2-dimensional object, you specify coordinates using the x-and y-axes as follows:

- You use the x-axis to specify the horizontal position on the canvas.
- You use the y-axis to specify the vertical position on the canvas.

The origin point in Photoshop, that is, x-axis = 0 and y-axis = 0, is the upper left corner of the screen. The opposite corner, the lower right, is the extreme point of the canvas. For example, if your canvas is 1000 x 1000 pixels, then the coordinate for the lower right corner is x-axis = 1000 and y-axis = 1000.

You specify coordinate points that describe the shape you want to select as an array, which then becomes the argument or parameter value for the `select/Select/select()` command.

The following examples assume that the ruler units have been set to pixels and create a selection by:

1. Creating a variable to hold a new document that is 500 x 500 pixels in size.
2. Creating a variable to hold the coordinates that describe the selected area (that is, the `Selection` object).
3. Adding an array as the selection variable's value.
4. Using the `Document` object's `selection` property, and the `Selection` object's `select` command to select an area. The area's coordinates are the selection variable's values.

**AS**

```plaintext
set docRef to make new document with properties {height:500, width:500}
set shapeRef to {{0, 0}, {0, 100}, {100, 100}, {100, 0}}
select current document region shapeRef
```

**VBS**

```plaintext
DocRef = appRef.Documents.Add
ShapeRef = Array(Array(0, 0), Array(0, 100), Array(100, 100), Array(100,0))
docRef.Selection.Select ShapeRef
```
Working with the Photoshop Object Model

**Stroking the selection border**

The following examples use the `stroke` command of the `Selection` object to stroke the boundaries around the current selection and set the stroke color and width.

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

**AS**

```actionscript
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
```

**VBS**

```vbscript
Set strokeColor = CreateObject("Photoshop.SolidColor")
strokeColor.CMYK.Cyan = 20
strokeColor.CMYK.Magenta = 50
strokeColor.CMYK.Yellow = 30
strokeColor.CMYK.Black = 0
appRef.ActiveDocument.Selection.Stroke strokeColor, 5, 1, 15, 75, False
```

**JS**

```javascript
strokeColor = new solidColor
strokeColor.cmyk.cyan = 20
strokeColor.cmyk.magenta = 50
strokeColor.cmyk.yellow = 30
strokeColor.cmyk.black = 0

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

**Inverting selections**

You can use the `invert` command of the `Selection` object to a selection so you can work on the rest of the document, layer or channel while protecting the selection.

➢ **AS:** `invert selection of current document`
➢ **VBS:** `selRef.Invert`
➢ **JS:** `selRef.invert()`

**Expanding, contracting, and feathering selections**

You can change the size of a selected area using the expand, contract, and feather commands.

The values are passed in the ruler units stored in Photoshop preferences and can be changed by your scripts. If your ruler units are set to pixels, then the following examples will expand, contract, and feather
by five pixels. See section “Setting Application Preferences” on page 32 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

**VBS**
Dim selRef
Set selRef = appRef.ActiveDocument.Selection
selRef.Expand 5
selRef.Contract 5
selRef.Feather 5

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

### Filling a selection

You can fill a selection either with 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

**VBS**
Set fillColor = CreateObject("Photoshop.SolidColor")
fillColor.RGB.Red = 255
fillColor.RGB.Green = 0
fillColor.RGB.Blue = 0
selRef.Fill fillColor, 15, 25, 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 tenth item in the history state:**

**NOTE:** See "Using history state objects" on page 47 for information on History State objects.

**AS**
fill selection of current document with contents history state 10¬
of current document

**VBS**
selRef.Fill docRef.HistoryStates(10)

**JS**
selRef.fill(app.activeDocument.historyStates[9])

### Loading and storing selections

You can store Selection objects in, or load them from, Channel objects. To store a selection in a channel, it should have its kind (Kind/Kind) property set to a type that indicates that channel holds a selected area: selected area channel (psSelectedAreaAlphaChannel)/ChannelType.SELECTEDAREA). The following
examples use the store (Store/store()) command of the Selection object to store the current selection in a channel named My Channel and extend the selection with any selection that is currently in that channel.

AS
set myChannel to make new channel of current document with properties ¬
{name:"My Channel", kind::selected area channel}
store selection of current document into channel ¬
"My Channel" of current document combination type extended

VBS
Set chanRef = docRef.Channels.Add
chanRef.Name = "My Channel"
chanRef.Kind = 3 'psSelectedAreaAlphaChannel
docRef.Selection.Store docRef.Channels("My Channel"), 2
'PsSelectionType is 2 (psExtendSelection)

JS
var chanRef = docRef.channels.add()
chanRef.name = "My Channel"
chanRef.kind = ChannelType.SELECTEDAREA
docRef.selection.store(docRef.channels["My Channel"], SelectionType.EXTEND)

To restore a selection that has been saved to a Channel object, use the load (Load/load) method.

AS
set myChannel to make new channel of current document with properties ¬
{name:"My Channel"}
load selection of current document from channel "My Channel" of ¬
current document combination type extended

VBS
selRef.Load docRef.Channels("My Channel"), 2
'PsSelectionType is 2 (psExtendSelection)

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

See section “Understanding Clipboard Interaction” on page 54 for examples on how to copy, cut and paste selections.

Working with Channel objects

The Channel object 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. See “Creating New Objects in a Script” on page 23 for information on creating a Channel object in your script.

You can set or get (that is, find out about) a Channel object's type using the kind property. See “Loading and storing selections” on page 45 for script samples that demonstrate how to create a selected area channel.

Changing channel types

You can change the kind of any channel except component channels. The following examples demonstrate how to change a masked area channel to a selected area channel:

NOTE: Component channels are related to the document mode. Refer to Photoshop Help for information on channels, channel types, and document modes.
CHAPTER 3: Scripting Photoshop

Working with the Photoshop Object Model

AS  set kind of myChannel to selected area channel
VBS  channelRef.ind = 3 'for psSelectedAreaAlphaChannel
     'from the constant value PsChannelType
JS   channelRef.kind = ChannelType.SELECTEDAREA

Using the Document Info object

In Photoshop, you can associate information with a document by choosing File > File Info.

To accomplish this task in a script, you use the info-object (DocumentInfo/DocumentInfo) object, which is stored in the info (Info/info) property of the Document object. The following examples demonstrate how to use the DocumentInfo object to set the copyrighted status and owner URL of a document.

AS  set docInfoRef to info of current document
    get EXIF of docInfoRef
    set copyrighted of docInfoRef to copyrighted work
    set owner url of docInfoRef to "http://www.adobe.com"
    get EXIF of docInfoRef
VBS  Set docInfoRef = docRef.Info
    docInfoRef.Copyrighted = 1 'for psCopyrightedWork
    docInfoRef.OwnerUrl = "http://www.adobe.com"
JS   docInfoRef = docRef.info
    docInfoRef.copyrighted = CopyrightedType.COPYRIGHTEDWORK
    docInfoRef.ownerUrl = "http://www.adobe.com"

For information about other types of information (properties) you can associate with a document, look up the following:

➤ In the Adobe Photoshop CS6 AppleScript Scripting Reference or in the Photoshop AppleScript Dictionary, look up the properties for the class info-object.

➤ In the Adobe Photoshop CS6 Visual Basic Scripting Reference, the Adobe Photoshop CS6 JavaScript Scripting Reference, the Visual Basic Object Browser or the ExtendScript Object Model Viewer, look up the properties for the DocumentInfo object.

Using history state objects

Photoshop keeps a history of the actions that affect documents. Each time you apply a change to an image in the Photoshop application, you create a history state; you can access a document's history states from the History palette by selecting Window > History. See Photoshop Help for additional information about History State.

In a script, you can access a Document object's history states using the HistoryStates object, which is a property of the Document object. You can use a HistoryStates object to reset a document to a previous state or to fill a Selection object.

The following examples revert the document contained in the variable docRef back to the form and properties it had when it was first opened or created. Using history states in this fashion gives you the ability to undo modifications to the document.
CHAPTER 3: Scripting Photoshop

Working with the Photoshop Object Model

AS

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

VBS
docRef.ActiveHistoryState = docRef.HistoryStates(1)

JS
docRef.activeHistoryState = docRef.historyStates[0]

**NOTE:** Reverting back to a previous history state does not remove any later states from the history collection. Use the **Purge** command to remove later states from the **History States** collection as shown below:

➤ **AS:** purge history caches
➤ **VBS:** appRef.Purge(2) 'for psPurgeTarget -- > 2 (psHistoryCaches)
➤ **JS:** app.purge(PurgeTarget.HISTORYCACHES)

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 layer of current document using motion blur with options ~
{class:motion blur, angle:20, radius:20}
set current history state of current document to savedState

VBS

Set savedState = docRef.ActiveHistoryState
docRef.ArtLayers(1).ApplyMotionBlur 20, 20
docRef.ActiveHistoryState = savedState

JS

savedState = docRef.activeHistoryState
docRef.artLayers[0].applyMotionBlur( 20, 20 )
docRef.activeHistoryState = savedState

Using Notifier objects

You use the **Notifier** object to tie an event to a script. For example, if you would like Photoshop to automatically create a new document when you open the application, you could tie a script that creates a **Document** object to an **Open Application** event.

**NOTE:** This type of script corresponds to selecting **Start Application** in the Script Events Manager (**File > Scripts > Script Events Manager**) in the Photoshop application. Please refer to Photoshop Help for information on using the Script Events Manager.

The **make (Add/add)** command requires you to specify an event ID to identify the event to set up notification for. Many event IDs are listed in an Appendix in the Adobe Photoshop CS6 JavaScript Scripting Reference, Adobe Photoshop CS6 Visual Basic Scripting Reference, and Adobe Photoshop CS6 AppleScript Scripting Reference. Some events also operate on several types of objects, and the **make (Add/add)** command requires an additional argument for a class ID, which identifies the object. For example, the “New” command is used for Document, Art Layer, and Channel objects.

**NOTE:** You can determine the event and class IDs of any recordable event by using ScriptListener. See “Using ScriptListener to find event IDs and class IDs” on page 81.

The following example shows how to set up event notification for an “Open Document” event. First the script ensures that event notification is enabled, then it sets up the event to trigger the execution of the Welcome.jsx file. Once the script completes, any time you open a document outside of a script, it triggers the notification, which runs the .jsx file. This .jsx file displays an alert box.
NOTE: Notification generally does not take effect on events that occur inside of a script, because these events are embedded with in an "AdobeScriptAutomation Scripts" event.

**AS**

tell application "Adobe Photoshop CS6"
    try
        delete notifiers
    end try
    make new notifier with properties {event:"Opn \", ¬
        event file:alias "OS X 10.5.8 US:Users:psauto:Desktop:Welcome.jsx"}
end tell

**VBS**

Dim appRef,eventFile
Set appRef = CreateObject("Photoshop.Application")
appRef.NotifiersEnabled = True
eventFile = appRef.Path & "Presets\Scripts\Event Scripts Only\Welcome.jsx"
appRef.Notifiers.Add "Opn \", eventFile

**JS**

app.notifiersEnabled = true
var eventFile = new File(app.path +
    "/Presets/Scripts/Event Scripts Only/Welcome.jsx")
app.notifiers.add("Opn \", eventFile)

Using the PathItem object

To create a PathItem object, you must add a PathItem to the PathItems element or collection for a document. This requires that you first create an array of PathPointInfo objects, which specify the coordinates of the corners or anchor points of your path. Then you create an array of SubPathInfo objects to contain the PathPoint arrays. Once you have the points and a subpath, you can add a new PathItem.

The following script creates a PathItem object that is a straight line.

**AS**

--line #1--it’s a straight line so the coordinates for anchor, left, and
--right for each point have the same coordinates
tell application "Adobe Photoshop CS6"
    set ruler units of settings to pixel units
    set type units of settings to pixel units
    set docRef to make new document with properties {height:700, width:500, ¬
        name:"Snow Cone"}

    set pathPointInfo1 to {class:path point info, kind:corner point, ¬
        anchor:{100, 100}, left direction:{100, 100}, right direction:{100, 100}}
    set pathPointInfo2 to {class:path point info, kind:corner point, ¬
        anchor:{150, 200}, left direction:{150, 200}, right direction:{150, 200}}
    set subPathInfo1 to ¬
        {class:sub path info, ¬
            entire sub path:[pathPointInfo1, pathPointInfo2], ¬
            operation:shape xor, closed:false}

    set newPathItem to make new path item in docRef with properties ¬
        {entire path:{subPathInfo1}, name:"Line", kind:normal}
end tell
VBS

```vbs
Dim appRef, docRef
Dim lineArray(1), lineArray2(1), lineSubPathArray(0), myPathItem
Set appRef = CreateObject("Photoshop.Application")

' create a document to work with
Set docRef = appRef.Documents.Add(5000, 7000, 72, "Simple Line")

'line #1--it's a straight line so the coordinates for anchor, left, and 'right for each point have the same coordinates
'First create the array of PathPointInfo objects. The line has two points, 'so there are two PathPointInfo objects.
Set lineArray(0) = CreateObject("Photoshop.PathPointInfo")
lineArray(0).Kind = 2 ' for PsPointKind --> 2 (psCornerPoint)
lineArray(0).Anchor = Array(100, 100)
lineArray(0).LeftDirection = lineArray(0).Anchor
lineArray(0).RightDirection = lineArray(0).Anchor
Set lineArray(1) = CreateObject("Photoshop.PathPointInfo")
lineArray(1).Kind = 2
lineArray(1).Anchor = Array(150, 200)
lineArray(1).LeftDirection = lineArray(1).Anchor
lineArray(1).RightDirection = lineArray(1).Anchor

'Next create a SubPathInfo object, which will hold the line array 'in its EntireSubPath property.
Set lineSubPathArray(0) = CreateObject("Photoshop.SubPathInfo")
lineSubPathArray(0).Operation = 2 ' for PsShapeOperation --> 2 (psShapeXOR)
lineSubPathArray(0).Closed = false
lineSubPathArray(0).EntireSubPath = lineArray

'create the PathItem object using Add. This method takes the SubPathInfo object 'and returns a PathItem object, which is added to the pathItems collection 'for the document.
Set myPathItem = docRef.PathItems.Add("A Line", lineSubPathArray)

' stroke it so we can see something
myPathItem.StrokePath(2) ' for PsToolType --> 2 (psBrush)
```

JS

```javascript
var docRef = app.documents.add(5000, 7000, 72, "Simple Line")

// create a document to work with
var docRef = app.documents.add(5000, 7000, 72, "Simple Line")

//line #1--it's a straight line so the coordinates for anchor, left, and //right //for each point have the same coordinates
// First create the array of PathPointInfo objects. The line has two points, // so there are two PathPointInfo objects.
var lineArray = new Array()
  lineArray[0] = new PathPointInfo
  lineArray[0].kind = PointKind.CORNERPOINT
  lineArray[0].anchor = Array(100, 100)
  lineArray[0].leftDirection = lineArray[0].anchor
  lineArray[0].rightDirection = lineArray[0].anchor
  lineArray[1] = new PathPointInfo
  lineArray[1].kind = PointKind.CORNERPOINT
  lineArray[1].anchor = Array(150, 200)
  lineArray[1].leftDirection = lineArray[1].anchor
  lineArray[1].rightDirection = lineArray[1].anchor
```
// Next create a SubPathInfo object, which holds the line array
// in its entireSubPath property.
var lineSubPathArray = new Array()
    lineSubPathArray[0] = new SubPathInfo()
    lineSubPathArray[0].operation = ShapeOperation.SHAPEXOR
    lineSubPathArray[0].closed = false
    lineSubPathArray[0].entireSubPath = lineArray

// create the path item, using add. This method takes the SubPathInfo object
// and returns a PathItem object, which is added to the pathItems collection
// for the document.
var myPathItem = docRef.pathItems.add("A Line", lineSubPathArray);

// stroke it so we can see something
myPathItem.strokePath(ToolType.BRUSH)

Working with color objects

Your scripts can use the same range of colors that are available from the Photoshop user interface. Each color model has its own set of properties. For example, the RGB color (RGBColor/RGBColored) class contains three properties: red, blue and green. To set a color in this class, you indicate values for each of the three properties.

In VBScript and JavaScript, the SolidColor class contains a property for each color model. To use this object, you first create an instance of a SolidColor object, then set appropriate color model properties for the object. Once a color model has been assigned to a SolidColor object, the SolidColor object cannot be reassigned to a different color model.

The following examples demonstrate how to set a color using the CMYK color class.

**AS**

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

**VBS**

```vbs
'create a solidColor array
Dim solidColorRef
Set solidColorRef = CreateObject("Photoshop.SolidColor")
solidColorRef.CMYK.Cyan = 20
solidColorRef.CMYK.Magenta = 90
solidColorRef.CMYK.Yellow = 50
solidColorRef.CMYK.Black = 50
appRef.ForegroundColor = solidColorRef
```

**JS**

```js
//create a solid color array
var solidColorRef = new solidColor()
solidColorRef.cmyk.cyan = 20
solidColorRef.cmyk.magenta = 90
solidColorRef.cmyk.yellow = 50
solidColorRef.cmyk.black = 50
foregroundColor = solidColorRef
```
**Solid color classes**

The solid color classes available in Photoshop are illustrated below.

### Color Classes

- Solid Color
- RGB Color
- CMYK Color
- Gray Color
- HSB Color
- Lab Color
- No Color

**Using hex values for RGB color**

You can express RGB colors as hex (or hexadecimal) values. A hex value contains three pairs of numbers which represent red, blue and green (in that order).

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 VBScript and JavaScript, the RGBColor object has a string property called HexValue/hexValue.

**Getting and converting colors**

The following examples convert an RGB color to its CMYK equivalent.

**AS**

The following script, which assumes an RGB color model, gets the foreground color and then uses the convert command of the color class to convert the color to its CMYK equivalent.

```applescript
get foreground color
convert color foreground color to CMYK
```

Look up the following in the Adobe Photoshop CS6 AppleScript Scripting Reference or in the Photoshop AppleScript Dictionary:

- In the “Objects” section, the foreground color property of the class application
- In the “Commands” section, convert

**VBS**

The following script uses an If Then statement and the model property of the SolidColor object to determine the color model in use. The If Then statement returns a SolidColor object; if it returns an RGB object, the cmyk property of the SolidColor object then allows you to access the color with its CMYK equivalent.

```vbscript
Dim someColor
If (someColor.model = 2) Then
    someColor.cmyk
    'someColor.model = 2 indicates psColorModel --> 2 (psRGBModel)
End If
```
Look up the following in the *Adobe Photoshop CS6 Visual Basic Scripting Reference*, or in the Visual Basic Object Browser:

- *model* and *cmyk* as properties of the *SolidColor* object

**JS**
This example uses the *foregroundColor* property of the *Application* object to get the original color to be converted. The *cmyk* property of the *SolidColor* object that *foregroundColor* refers to provides a way to access the cmyk equivalent of the rgb color.

```javascript
var someColor = foregroundColor.cmyk
```

Look up the following in the *Adobe Photoshop CS6 JavaScript Scripting Reference*, or in the ExtendScript Object Model Viewer:

- *cmyk* as a property of the *SolidColor* object
- *foregroundColor* as a property of the *Application* object

**Comparing colors**

Using the *equal colors* (*isEqual/isEqual*) command, you can compare colors. The following statements return true if the foreground color is visually equal to background color.

- **AS:**
  ```app Ref . ForegroundColor = isEqual (appRef.BackgroundColor)```
- **VBS:**
  ```If (appRef.ForegroundColor.IsEqual(appRef.BackgroundColor)) Then```
- **JS:**
  ```if (app . foregroundColor . isEqual (backgroundColor))```

**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 of the *SolidColor* object for VBScript and JavaScript.

- **AS:**
  ```set myWebSafeColor to web safe color for foreground color```
- **VBS:**
  ```Dim myWebSafeColor
  Set myWebSafeColor = appRef.ForegroundColor.NearestWebColor```
- **JS:**
  ```var webSafeColor = new RGBColor()
  webSafeColor = app.foregroundColor.nearestWebColor```

**Working with Filters**

To apply a filter in AppleScript, you use the *filter* command with an option from the class *filter options*. In VBScript and JavaScript, you use a specific filter method. For example, to apply a Gaussian blur filter, you use the *ApplyGaussianBlur/applyGaussianBlur()* method. All filter methods belong to the *ArtLayer* object.

**NOTE:** Please refer to Photoshop Help for information about the effects produced by individual filter types.

The following examples apply the Gaussian blur filter to the active layer.
Use the `filter` command and then both specify the layer and the name of the filter and any options.

```plaintext
filter current layer of current document using gaussian blur
    with options {radius:5}
```

**NOTE:** In the *Adobe Photoshop CS6 AppleScript Scripting Reference*, or in the Photoshop AppleScript Dictionary, look up the `filter` command; also look up class `filter` options.

```plaintext
VBS
appRef.docRef.ActiveLayer.ApplyGaussianBlur 5
```

**NOTE:** In the *Adobe Photoshop CS6 Visual Basic Scripting Reference*, or in the Visual Basic Object Browser look up the `ApplyGaussianBlur` method and other methods of the `ArtLayer` object whose name begins with “Apply.”

```plaintext
JS
docRef.activeLayer.applyGaussianBlur(5)
```

**NOTE:** In the *Adobe Photoshop CS6 JavaScript Scripting Reference*, or in the ExtendScript Object Model Viewer look up the `applyGaussianBlur()` method and other methods of the `artLayer` object whose name begins with “apply.”

### 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 or VBScript, you can run the JavaScript from your script. See “Action Manager” on page 73 for information on using the Action Manager. Also, see “Executing JavaScripts from AS or VBS” on page 10.

### Understanding Clipboard Interaction

The clipboard commands in Photoshop operate on `ArtLayer`, `Selection`, and `Document` objects. The commands can be used to operate on objects within a single document, or to move information between documents.

The clipboard commands of the `art layer` (`ArtLayer`/`ArtLayer`) and `selection` (`Selection`/`Selection`) objects are:

- copy (Copy/copy)
- copy merged (Copy Merge parameter value/copy(merge parameter value))
- cut (Cut/cut)

The clipboard commands of the `document`/`Document`/`Document` object are:

- paste (Paste/paste)
- paste into (Paste IntoSelection parameter value/paste(intoSelection parameter value))

**NOTE:** For information on copy, copy merged, paste, paste into, and cut functions, see Photoshop Help.

### Using the copy and paste commands

The following examples copy the contents of the background layer to the clipboard, create a new document, and then paste the clipboard contents to the new document. The scripts assume that there is a document already open in Photoshop and that the document has a background layer.
NOTE: If your script creates a new document in which you paste the clipboard contents, be sure the document uses the same ruler units as the original document. See “Setting Application Preferences” on page 32 for information.

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

tell application "Adobe Photoshop CS6"
    activate
    select all of current document
    copy
    set current layer of current document to layer "Background" of current document
    set newDocRef to make new document
    paste newDocRef
end tell

VBS
'make firstDocument the active document
Set docRef = appRef.ActiveDocument
docRef.ArtLayers("Background").Copy

Set newDocRef = appRef.Documents.Add(8, 6, 72, "New Doc")
newDocRef.Paste

JS
//make firstDocument the active document
var docRef = app.activeDocument
docRef.artLayers["Background"].copy()

var newDocRef = app.documents.add(8, 6, 72, "New Doc")
newDocRef.paste()

Using the copy merged command/method

You can also perform a merged copy to copy all visible layers in the selected area. In AppleScript, you use the copy merged command. For VBScript and JavaScript, you use the Copy/copy command, passing in a value of True/true for the optional merge parameter.

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

set docRef to make new document
make new art layer of docRef
select all of docRef
copy merged selection of docRef

VBS
docRef.Selection.Copy True

Look up the Copy method for the ArtLayer and Selection objects in the Adobe Photoshop CS6 Visual Basic Scripting Reference, or in the Visual Basic Object Browser

JS
docRef.selection.copy(true)

Look up the copy() method for the ArtLayer and Selection objects in the Adobe Photoshop CS6 JavaScript Scripting Reference, or in the ExtendScript Object Model Viewer.
Working with Units

Photoshop provides two rulers for documents. Using properties on the `settings-object` (Preferences/Preferences) object, you can set the measurement units for the rulers in your script. The rulers are:

- A graphics ruler used for most graphical layout measurements or operations on a document where height, width, or position are specified.
  
  You set measurement unit types for the graphics ruler using the `ruler units (RulerUnits/rulerUnits)` property.

- A type ruler, which is active when using the type tool.
  
  You set measurement unit types for the type ruler using the `type units (TypeUnits/typeUnits)` property.

**Note:** These settings correspond to those found in the Photoshop preference dialog under Photoshop > Preferences > Units & Rulers on Mac OS or Edit > Preferences > Units & Rulers in Windows.

Unit values

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 VBScript statement sets a document’s size to 3 inches by 3 inches:

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

If the ruler units had been set to pixels, the document would be 3 pixels by 3 pixels. 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 ruler settings should be restored if changed in the script. See “Setting ruler and type units in a script” on page 59 for directions on setting unit values.

Please refer to Photoshop Help for information about available unit value types.

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.

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:

```applescript
make new document with properties {width:inches 4, ¬
    height:inches 5}
```
The values returned for a Photoshop property that uses units is returned as a value of the current ruler type. Getting the height of the document created above:

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

returns 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 returns a value of 360 (5 inches x 72 points per inch).

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.

You can convert, or coerce, a unit value from one value type to another. For example, the following script converts a point value to an inch value.

```
set pointValue to 72 as points
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.

**NOTE:** The unit values **cm units** and **mm units** cannot be used in this way with a corresponding reference to **cm** or **mm**. They are not supported by the AppleScript terminology.

### Using unit values in calculations

To use a unit value in a calculation in AppleScript 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) * 5
```

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

**NOTE:** 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 29.99 if your document resolution is set to 72 ppi. The scripting interface assumes settings are measured by ppi.

### Unit value usage

The following tables list the properties of the classes/objects that are defined to use unit values. Unit values for these properties, unless otherwise indicated in the table, are based the graphics ruler setting.
To use this table, do one of the following:

➤ Look up the properties of the class in the *Adobe Photoshop CS6 AppleScript Scripting Reference*, or in the Photoshop AppleScript Dictionary.

➤ Look up the property of the object in the *Adobe Photoshop CS6 Visual Basic Scripting Reference*, the *Adobe Photoshop CS6 JavaScript Scripting Reference*, the Visual Basic Object Browser, or the ExtendScript Object Model Viewer.

<table>
<thead>
<tr>
<th>Class/object</th>
<th>AppleScript properties</th>
<th>VBScript properties</th>
<th>JavaScript properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>height</td>
<td>Height</td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>Width</td>
<td>width</td>
</tr>
<tr>
<td>EPS open options</td>
<td>height</td>
<td>Height</td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>Width</td>
<td>width</td>
</tr>
<tr>
<td>PDF open options</td>
<td>height</td>
<td>Height</td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>Width</td>
<td>width</td>
</tr>
<tr>
<td>lens flare open options</td>
<td>height</td>
<td>Height</td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>Width</td>
<td>width</td>
</tr>
<tr>
<td>offset filter</td>
<td>horizontal offset</td>
<td>HorizontalOffset</td>
<td>horizontalOffset</td>
</tr>
<tr>
<td></td>
<td>vertical offset</td>
<td>VerticalOffset</td>
<td>verticalOffset</td>
</tr>
<tr>
<td>Text Item</td>
<td>baseline shift*</td>
<td>BaselineShift*</td>
<td>baselineShift*</td>
</tr>
<tr>
<td></td>
<td>first line indent*</td>
<td>FirstLineIndent*</td>
<td>firstLineIndent*</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>Height</td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>hyphenation zone*</td>
<td>HyphenationZone*</td>
<td>hyphenationZone*</td>
</tr>
<tr>
<td></td>
<td>leading*</td>
<td>Leading*</td>
<td>leading*</td>
</tr>
<tr>
<td></td>
<td>left indent*</td>
<td>LeftIndent*</td>
<td>leftIndent*</td>
</tr>
<tr>
<td></td>
<td>position</td>
<td>Position</td>
<td>position</td>
</tr>
<tr>
<td></td>
<td>right indent*</td>
<td>RightIndent*</td>
<td>rightIndent*</td>
</tr>
<tr>
<td></td>
<td>space before*</td>
<td>SpaceBefore*</td>
<td>spaceBefore*</td>
</tr>
<tr>
<td></td>
<td>space after*</td>
<td>SpaceAfter*</td>
<td>spaceAfter*</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>Width</td>
<td>width</td>
</tr>
</tbody>
</table>

* Unit values based on type ruler setting.

The following table lists the commands that use unit values as parameters or arguments. In some cases the parameters are required. The VBScript and JavaScript methods are preceded by the object to which they belong.

To use this table:

➤ For AppleScript commands, look up the command in the “Commands” chapter of the *Adobe Photoshop CS6 AppleScript Scripting Reference*, or use the Photoshop AppleScript Dictionary.

➤ For VBScript methods, look up the method in the Methods table of the object in the “Interface” chapter of the *Adobe Photoshop CS6 Visual Basic Scripting Reference*, or use the Visual Basic Object Browser.

➤ For JavaScript methods, look up the method in the Methods table of the object in the “Object Reference” chapter in the *Adobe Photoshop CS6 JavaScript Scripting Reference*, or use the ExtendScript Object Model Viewer.
### Setting ruler and type units in a script

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.

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

```appRef.Preferences.RulerUnits = 2  
appRef.Preferences.TypeUnits = 1  
appRef.Preferences.PointSize = 2```

AS indicates that `2` represents `PsUnitType --> 1` (PsInches).

**VBS**  
In VBScript, `RulerUnits` and `TypeUnits` are properties of the `Preferences object`, accessed through the `Application object's preferences property` as shown below.

```appRef.Preferences.RulerUnits = 2  
appRef.Preferences.TypeUnits = 1  
appRef.Preferences.PointSize = 2
```

VBS indicates that `2` represents `PsUnitType --> 1` (PsInches).

**JS**  
In JavaScript, `rulerUnits` and `typeUnits` are properties of the `Preferences object`, accessed through the `Application object's preferences property` as shown below.

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

**NOTE:** Remember to reset the unit settings back to the original values at the end of a script. See “Working with document preferences” on page 61 for an example of how to do this.
Sample Workflow Automation JavaScripts

The following sample workflow automation JavaScripts are provided with Photoshop and demonstrate various kinds of scripting usage. The scripts are located in the Presets/Scripts folder in your application directory. See “Creating and running a JavaScript" on page 19 for information on the Presets/Scripts folder.

<table>
<thead>
<tr>
<th>Script name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer Comps to Files.jsx</td>
<td>Saves layer comp as files.</td>
</tr>
<tr>
<td>Layer Comps to PDF.jsx</td>
<td>Saves layer comp as a PDF presentation.</td>
</tr>
<tr>
<td>Layer Comps to WPG.jsx</td>
<td>Saves layer comp as a Web photo gallery.</td>
</tr>
<tr>
<td>Export Layers to Files.jsx</td>
<td>Exports each layer in the document to a separate file.</td>
</tr>
<tr>
<td>Script Events Manager.jsx</td>
<td>Enables and disables notifier objects.</td>
</tr>
<tr>
<td>Image Processor.jsx</td>
<td>Processes camera raw images into various file formats.</td>
</tr>
<tr>
<td>Load Files into Stack.jsx</td>
<td>Loads separate files into an image stack in a single document.</td>
</tr>
<tr>
<td>Merge to HDR.jsx</td>
<td>Combines multiple images of the same scene or image, capturing the dynamic range of a scene in a single High Dynamic Range (HDR) image.</td>
</tr>
</tbody>
</table>

Advanced Scripting

This section demonstrates how to use the information contained in the previous sections of this chapter to create scripts that do the following:

➤ Configure document preferences.

➤ Apply color to text items. In this section, you will also learn how to do the following:

  ➤ Create a reference to an existing document.

  ➤ Create a layer object and make the layer a text layer.

➤ Rasterize text so that wave and blur processing can be applied to words. In these sections you will also learn how to do the following:

  ➤ Select and work with a specific area of a layer by creating a selection object.

  ➤ Apply wave and motion blur filters to selected text.

**NOTE:** When you finish the lesson in each of the following sections, save the script you have created in the lesson. Each lesson builds upon the script created in the previous lesson.
Working with document preferences

The sample scripts in this section activate a Photoshop Application object and then save the default configuration settings into variables so that they can be restored later when the script completes. These are the default configurations you probably set up in the Preferences dialog when you initially installed and configured Photoshop.

**NOTE:** To view or set the Preferences on Mac OS, choose Photoshop > Preferences > Units & Rulers; in Windows choose Edit > Preferences > Units & Rulers.

Next, the scripts set the following preferences to the following values:

<table>
<thead>
<tr>
<th>Preference</th>
<th>Set to</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>rulers</td>
<td>inches</td>
<td>Uses inches as the unit of measurement for graphics.</td>
</tr>
<tr>
<td>units</td>
<td>pixels</td>
<td>Uses pixels as the unit of measurement for text (type).</td>
</tr>
<tr>
<td>dialog modes</td>
<td>never</td>
<td>Suppresses the use of dialogs so that your script executes without the user being asked for input (such as clicking an OK button) at various stages of the process.</td>
</tr>
</tbody>
</table>

**NOTE:** dialog modes is not an option in the Photoshop application.

Next, the script declares variables that store document dimensions in inches and document resolution in pixels. The script then declares a display resolution, and assigns the text "Hello, World!" to a string variable.

Next, an `if` statement checks whether a `Document` object has been created and then creates a new `Document` object if none exists.

Finally, the script restores the original preferences.

**AS**

To work with document preferences:

1. Create and run the following script. See “Creating and running an AppleScript” on page 18 for details.

```applescript
tell application "Adobe Photoshop CS6"
    --make Photoshop CS6 the active (front-most) application
    activate

    --create variables for the default settings
    set theStartRulerUnits to ruler units of settings
    set theStartTypeUnits to type units of settings
    set theStartDisplayDialogs to display dialogs

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

    --create variables for default document settings
    set theDocWidthInInches to 4
    set theDocHeightInInches to 2
    set theDocResolution to 72
    set theDocString to "Hello, World!"
```
--check to see whether any documents are open
--if none are found, create a document
--use the default document settings as its properties
if (count of documents) is 0 then
    make new document with properties ¬
    {width:theDocWidthInInches, height:theDocHeightInInches,¬
     resolution:theDocResolution, name:theDocString}
end if

--change the settings back to the original units stored in the variables
set ruler units of settings to theStartRulerUnits
set type units of settings to theStartTypeUnits
set display dialogs to theStartDisplayDialogs
end tell

2. In Photoshop, choose Photoshop > Preferences > Units & Rulers to verify that your preferences have been returned to your original settings.

3. After viewing the document in Photoshop, close the document without saving it.

4. Save the script as HelloWorldDoc.

VBS

To work with document preferences:

1. Create the following script. See “Creating and running a VBScript” on page 19 for details.

   'create variables for default preferences, new preferences
   Dim startRulerUnits
   Dim startTypeUnits
   Dim docWidthInInches
   Dim docHeightInInches
   Dim resolution
   Dim helloWorldStr
   Dim appRef

   Set appRef = CreateObject("Photoshop.Application")

   'assign default preferences to save values in variables
   startRulerUnits = appRef.Preferences.RulerUnits
   startTypeUnits = appRef.Preferences.TypeUnits
   startDisplayDialogs = appRef.DisplayDialogs

   'set new preferences and document defaults
   appRef.Preferences.RulerUnits = 2 'for PsUnits --> 2 (psInches)
   appRef.Preferences.TypeUnits = 1 'for PsTypeUnits --> 1 (psPixels)
   appRef.DisplayDialogs = 3 'for PsDialogModes --> 3 (psDisplayNoDialogs)
   docWidthInInches = 4
   docHeightInInches = 2
   resolution = 72
   helloWorldStr = "Hello, World!"

   'see if any documents are open
   'if none, create one using document defaults
   If appRef.Documents.Count = 0 Then
     appRef.Documents.Add docWidthInInches, docHeightInInches, resolution,
     helloWorldStr
     End If
CHAPTER 3: Scripting Photoshop

Advanced Scripting

To work with document preferences:

1. Create the following script.

```
//create and assign variables for default preferences
startRulerUnits = app.preferences.rulerUnits
startTypeUnits = app.preferences.typeUnits
startDisplayDialogs = app.displayDialogs

//change settings
app.preferences.rulerUnits = Units.INCHES
app.preferences.typeUnits = TypeUnits.PIXELS
app.displayDialogs = DialogModes.NO

//create and assign variables for document settings
docWidthInInches = 4
docHeightInInches = 2
resolution = 72
docName = "Hello World"

//use the length property of the documents object to
//find out if any documents are open
//if none are found, add a document
if (app.documents.length == 0)
    app.documents.add(docWidthInInches, docHeightInInches, resolution, docName)

//restore beginning preferences
app.preferences.rulerunits = startRulerUnits
app.preferences.typeunits = startTypeUnits
app.displayDialogs = startDisplayDialogs
```

2. Name the script HelloWorldDoc.jsx and save it in the Presets/Scripts folder.

3. Open Photoshop and choose File > Scripts > HelloWorldDoc to run the script.

4. Choose Edit > Preferences > Units & Rulers to verify that your preferences have been returned to your original settings.

5. After viewing the document in Photoshop, close the document without saving it.

6. Save the script.
Applying color to a text item

In this section, we will add a layer to the HelloWorldDoc script, then change the layer to a text object that displays the text Hello, World! in red.

Before you begin, do the following:

➤ Make sure Photoshop is closed.
➤ Open the script file HelloWorldDoc in your script editor application.

AS To create and specify details in a text item:

1. Type the following code into the HelloWorldDoc script immediately before the statements at the end of the file that restore original preferences.

   ```applescript
   --create a variable named theDocRef
   --assign the current (active) document to it
   set theDocRef to the current document

   --create a variable that contains a color object of the RGB color class
   --whose color is red
   set theTextColor to {class:RGB color, red:255, green:0, blue:0}

   --create a variable for the text layer, create the layer as an art layer object
   --and use the kind property of the art layer object to make it a text layer
   set theTextLayer to make new art layer in theDocRef with ¬
   properties {kind:text layer}

   --Set the contents, size, position and color of the 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 as inches, 1 as inches}
   set stroke color of text object of theTextLayer to theTextColor
   ```

2. Run the complete script. Be patient while Photoshop executes your commands one by one.

3. After viewing the document in Photoshop, close the document without saving it.

NOTE: Look up the following classes in the Adobe Photoshop CS6 AppleScript Scripting Reference or in the Photoshop AppleScript Dictionary to see if you understand how you used them in this script:

➤ RGB color class
➤ art layer class
To create and specify details in a text item:

1. Type the following code into the HelloWorldDoc script immediately before the statements at the end of the file that restore original preferences.

```
' create a reference to the active (current) document
Set docRef = appRef.ActiveDocument

' create a variable named textColor
' create a SolidColor object whose color is red
' assign the object to textColor
Set textColor = CreateObject ("Photoshop.SolidColor")
textColor.RGB.Red = 255
textColor.RGB.Green = 0
textColor.RGB.Blue = 0

' create an art layer object using the
' Add method of the ArtLayers class
' assign the layer to the variable newTextLayer
Set newTextLayer = docRef.ArtLayers.Add()

' use the Kind property of the Art Layers class to
' make the layer a text layer
newTextLayer.Kind = 2
newTextLayer.TextItem.Contents = helloWorldStr
newTextLayer.TextItem.Position = Array(0.75, 1)
newTextLayer.TextItem.Size = 36
newTextLayer.TextItem.Color = textColor
```

2. Run the complete script. Be patient while Photoshop executes your commands one by one.

3. After viewing the document in Photoshop, close the document without saving it.

**NOTE:** Look up the following classes in the Adobe Photoshop CS6 Visual Basic Scripting Reference, or in the Visual Basic Object Browser to see if you understand how you used them in this script:

- SolidColor
- ArtLayer
To create and specify details in a text item:

1. Type the following code into the HelloWorldDoc script immediately before the statements at the end of the file that restore original preferences.

```javascript
// create a reference to the active document
docRef = app.activeDocument

// create a variable named textColor
// create a SolidColor object whose color is red
// assign the object to textColor
textColor = new solidColor
textColor.rgb.red = 255
textColor.rgb.green = 0
textColor.rgb.blue = 0

helloWorldText = "Hello, World!"

// create a variable named newTextLayer
// use the add() method of the artLayers class to create a layer object
// assign the object to newTextLayer
newTextLayer = docRef.artLayers.add()

// use the kind property of the artLayer class to make the layer a text layer
newTextLayer.kind = LayerKind.TEXT
newTextLayer.textItem.contents = helloWorldText
newTextLayer.textItem.position = Array(0.75, 1)
newTextLayer.textItem.size = 36
newTextLayer.textItem.color = textColor
```

2. Save the script, and then open Photoshop and select the script from the Scripts menu (choose File > Script > HelloWorldDoc). Be patient while Photoshop executes your commands one by one.

3. After viewing the document in Photoshop, close Photoshop without saving the document.

**NOTE:** Look up the following classes in the Adobe Photoshop CS6 JavaScript Scripting Reference, or in the ExtendScript Object Model Viewer to see if you understand how you used them in this script:

- SolidColor
- ArtLayer. Notice that the `LayerKind.TEXT` value of the kind property uses the `LayerKind` constant. Constants are always depicted in upper case letters in Photoshop JavaScripts.

### Applying a wave filter

In this section we'll apply a wave filter to the word *Hello* in our document. This entails the following steps:

- Set the document width and height to pixels and then rasterize the text object in the Text Layer.

  **NOTE:** Because text is a vector graphic and cannot apply a wave filter to vector graphics, we must first convert the image to a bitmap. Rasterizing converts mathematically defined vector artwork to pixels. For more information on rasterizing, refer to Photoshop Help.

- Select the area of the layer to which we want to apply the wave filter.

  **NOTE:** See “Defining the area of a selection object” on page 67 in order to understand the code within the script that accomplishes this task.
➤ Apply a wave filter to the selection.

*NOTE:* The wave is a truncated sine curve.

**Defining the area of a selection object**

To define the area of a selection object, we create an array of coordinates, or points specified in pixels within the document. The array indicates the coordinates that define the outside corners of a rectangular area that begins at the top left corner of the document and extends half way across the document.

*NOTE:* You can define any number of points for a selected area. The number of coordinates determines the shape of the selection. The last coordinate defined must be the same as the first so that the area is a closed selection path.

*NOTE:* See “Photoshop Object Model” on page 11 for information on selection objects and other Photoshop objects.

The array values in order are:

➤ **Upper left corner of the selection:** 0, 0
   > 0 indicates the left-most column in the document.
   > 0 indicates the top row in the document.

➤ **Upper right corner:** theDocWidthInPixels / 2, 0
   > theDocWidthInPixels / 2 indicates the column in the middle of the document; that is, the column whose coordinate is the total number of columns in the document divided by 2.

   *NOTE:* The value of theDocWidthInPixels is the total number of pixels that defines the document’s horizontal dimension. Columns are arranged horizontally.

   > 0 indicates the top row in the document.

➤ **Lower right corner:** theDocWidthInPixels / 2, theDocHeightInPixels
   > theDocWidthInPixels / 2 indicates the middle of the document.

   > theDocHeightInPixels indicates the bottom row in the document; that is row whose coordinate is the total number of rows in the document.

   *NOTE:* The value of theDocHeightInPixels is the total number of pixels that determine the vertical dimension of the document. Rows are stacked vertically.

➤ **Lower left corner:** 0, theDocHeightInPixels
   > 0 indicates the left-most column in the document.

   > theDocHeightInPixels indicates the bottom row in the document.

➤ **Upper left corner of the selection:** 0, 0
   > This closes the selection path back at the initial point.
To select an area and apply a wave filter to it:

1. Type the following code into the script file HelloWorldDoc just above the statements that restore original preferences:

```
--create new variables to contain the document object’s width and height
--determine width and height values by multiplying the
--width and height in inches by the resolution
--(which equals the number of pixels per inch)
set theDocWidthInPixels to theDocWidthInInches * theDocResolution
set theDocHeightInPixels to theDocHeightInInches * theDocResolution

--use the rasterize command of the art layer object
rasterize theTextLayer affecting text contents

--create a variable named theSelRegion
--assign an array of coordinates as its value
set theSelRegion to {{0, 0}, ¬
    {theDocWidthInPixels / 2, 0}, ¬
    {theDocWidthInPixels / 2, theDocHeightInPixels}, ¬
    {0, theDocHeightInPixels}, ¬
    {0, 0}}

--replace the document object with the selection object
--so that the wave is applied only to the selected text
select theDocRef region theSelRegion combination type replaced

--apply the wave filter using the filter command of the
--wave filter class (inherited from the filter options super class)
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}
```

2. Choose Run to run the script.

3. After viewing the document in Photoshop, close the document without saving it.

4. Save the script in the Script Editor.

**NOTE:** Look up the following classes in the *Adobe Photoshop CS6 AppleScript Scripting Reference*, or in the Photoshop AppleScript Dictionary to see if you understand how you used them in this script:

- wave filter class
- art layer class: rasterize command, filter command
- document class: select command, combination type parameter
CHAPTER 3: Scripting Photoshop

Advanced Scripting

VBS To select an area and apply a wave filter to it:

1. Type the following code into the script file HelloWorldDoc just above the statements at the end of the file that restore original preferences:

   ' create new variables to contain doc width and height
   ' convert inches to pixels by multiplying the number of inches by
   ' the resolution (which equals number of pixels per inch)
   docWidthInPixels = docWidthInInches * resolution
   docHeightInPixels = docHeightInInches * resolution

   ' use the Rasterize() method of the ArtLayer class to
   ' convert the text in the ArtLayer object (contained in the newTextLayer variable)
   ' to postscript text type
   newTextLayer.Rasterize (1)

   ' create an array to define the selection property
   ' of the Document object
   ' define the selected area as an array of points in the document
   docRef.Selection.Select Array(Array(0, 0), _
                              Array(docWidthInPixels / 2, 0), _
                              Array(docWidthInPixels / 2, docHeightInPixels), _
                              Array(0, docHeightInPixels), Array(0, 0))

   ' use the ApplyWave() method of the ArtLayer class
   ' to apply the wave of the selected text
   newTextLayer.ApplyWave 1, 1, 100, 5, 10, 100, 100, 1, 1, 0

2. Double click the file name in Windows Explorer to run the script.

3. After viewing the document in Photoshop, close the document without saving it.

4. Save the script.

   NOTE: Look up the following classes in the Adobe Photoshop CS6 Visual Basic Scripting Reference, or in the Visual Basic Object Browser to see if you understand how you used them in this script:

   ➤ ArtLayer class: ApplyWave method, Rasterize method
   ➤ Selection class: Select method
To select an area and apply a wave filter to it:

1. Type the following code into the script file `HelloWorldDoc` just above the statements that restore original preferences:

```javascript
// create new variables to contain doc width and height
// convert inches to pixels by multiplying the number of inches by
// the resolution (which equals number of pixels per inch)
var docWidthInPixels = docWidthInInches * resolution;
var docHeightInPixels = docHeightInInches * resolution;

// use the rasterize method of the artLayer class
newTextLayer.rasterize(RasterizeType.TEXTCONTENTS);

// create a variable to contain the coordinate values
// for the selection object
var selRegion = Array([0, 0],
                      [docWidthInPixels / 2, 0],
                      [docWidthInPixels / 2, docHeightInPixels],
                      [0, docHeightInPixels],
                      [0, 0]);

// use the select method of the selection object
// to create an object and give it the selRegion values
// as coordinates
docRef.selection.select(selRegion);

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

2. Save the script, and then open Photoshop and select the script from the Scripts menu (choose File > Script > HelloWorldDoc).

3. After viewing the document in Photoshop, close Photoshop without saving the document.

**NOTE:** Look up the following classes in the *Adobe Photoshop CS6 JavaScript Scripting Reference*, or in the ExtendScript Object Model Viewer to see if you understand how you used them in this script:

- **ArtLayer**
  - `rasterize()` method. Notice that the `RasterizeType.TEXTCONTENTS` argument uses the `RasterizeType` constant. Constants are always depicted in upper case letters in Photoshop JavaScripts.
  - `applyWave()` method
Applying a MotionBlur filter

In this section, we will apply a different filter to the other half of our document.

**AS**

To apply a motionblur filter to HelloWorldDoc:

1. Type the following code into the script file HelloWorldDoc just above the statements that restore original preferences.

   ```applescript
   -- change the value of the variable theSelRegion
   -- to contain the opposite half of the screen
   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
   ```

2. Choose **Run** to run the script.

   **NOTE:** Look up the `motion blur` class in the *Adobe Photoshop CS6 AppleScript Scripting Reference*, or in the Photoshop AppleScript Dictionary to see if you understand how you used it in this script:

**VBS**

To apply a motionblur filter to HelloWorldDoc:

1. Type the following code into the script file HelloWorldDoc just above the statements that restore original preferences.

   ```vbscript
   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
   ```

2. Double click on the file in Windows Explorer to run the script.

   **NOTE:** Look up the `ArtLayer` class: `ApplyMotionBlur` method in the *Adobe Photoshop CS6 Visual Basic Scripting Reference*, or in the Visual Basic Object Browser to see if you understand how you used it in this script:
To apply a motionblur filter to HelloWorldDoc:

1. Type the following code into the script file HelloWorldDoc just above the statements that restore original preferences.

```javascript
// change the value of selRegion to the other half of the document
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();
```

2. Save the script, and then open Photoshop and select the script from the Scripts menu (choose File > Script > HelloWorldDoc).

**NOTE:** Look up the ArtLayer class applyMotionBlur() method in the Adobe Photoshop CS6 JavaScript Scripting Reference, or in the ExtendScript Object Model Viewer to see if you understand how you used it in this script.
Photoshop actions allow you to save time by automating repetitive tasks. You create and run actions in the application interface using the Actions palette.

You can also manage actions in scripts using a utility called the Action Manager. The Action Manager allows you to write scripts that target Photoshop functionality that is not otherwise accessible in the scripting interface, such as third party plug-ins and filters. The only requirement for using the Action Manager is that the task that you want to access from the Action Manager is recordable.

This chapter describes how to use the Action Manager and the scripting interface objects it includes.

The ScriptListener Plug-In

Before you use the Action Manager, you must install the ScriptListener plug-in. ScriptListener records a file with scripting code corresponding to the actions you perform in the UI.

**Tip:** Because ScriptListener records most of your actions, install ScriptListener only when you are creating Action Manager scripts. Leaving ScriptListener installed continuously will not only create large files that occupy memory on your hard drive, it can slow Photoshop performance.

When you perform a task or series of tasks in Photoshop, ScriptListener creates several files, which contain code that represents the actions taken in Photoshop:

- ScriptingListenerJS.log, containing JavaScript code,
- ScriptingListenerVB.log, containing VBScript code (Windows only).

ScriptListener creates these files on the desktop.

**Note:** There is no AppleScript interface to the Action Manager. However, you can access the Action Manager from an AppleScript by executing a JavaScript from AppleScript. See “Running JavaScript-based Action Manager code from AppleScript” on page 80.

Installing ScriptListener

The ScriptListener plug-in is located in the ..\Adobe Photoshop CS6\Scripting\Utilities folder.

**To install the ScriptListener:**

1. Select the file ScriptListener.8li and then choose Edit > Copy.
2. Paste the file copy to the following location:
   
   ..\Adobe Photoshop CS6\Plug-Ins\Automate

3. Open Photoshop.

   **Note:** If Photoshop is already open, close it and then start it again. This will allow Photoshop to load the plug-in.
CHAPTER 4: Action Manager

Action Manager Scripting Objects

To uninstall the ScriptListener:

1. Close Photoshop.
2. Verify that a copy of the file `ScriptListener.8li` still exists in the \`..\Adobe Photoshop CS6\Scripting\Utilities\` folder.
3. Delete the file `ScriptListener.8li` from the following location:
   \`..\Adobe Photoshop CS\Plug-Ins\Automate\`
4. Delete the log files `ScriptingListenerJS.log` and `ScriptingListenerVB.log` from your desktop.

**NOTE:** In Windows, even though you remove the ScriptListener from the Automate folder, it may continue to record actions. To prevent the `ScriptingListenerJS.log` file from becoming too large, delete it each time you finish playing a Photoshop action.

Action Manager Scripting Objects

The objects Action Descriptor, Action List, and Action Reference are part of the Action Manager functionality. For detailed information about these objects, see the appropriate reference manual, or use the object browser for the language you are using.

**NOTE:** These objects are not available in AppleScript.

Recording a Script using ScriptListener

The section demonstrates how to create a script log file using ScriptListener. We will record the actions necessary to apply the emboss filter to a document. (By default, the Emboss filter is available only via the Photoshop interface.)

**NOTE:** ScriptListener must be installed in the Automate folder before you begin the following procedure. See “Installing ScriptListener” on page 73.

To make the Emboss filter scriptable:

1. Open Photoshop, then open a document.
2. Choose Window > Actions, then choose New Action from the Actions palette menu.
3. Name the action, then click Record.
5. Using the following settings:
   - Angle: 135
   - Height: 3
   - Amount: 100
6. Click OK.
7. Check for the script log files:
   - In Windows, the log files are in your desktop.
   - On Mac OS, the log files are on the desktop.

Using the Action Manager from JavaScript

The section demonstrates how to use the contents of the `ScriptingListenerJS.log` log to create your script. Before you begin this section, you need have already recorded an action. The example in this section assumes you have followed the instructions in “Recording a Script using ScriptListener” on page 74.

The procedures in this section use the Action Manager to make the Emboss filter available to the scripting interface.

To create a JavaScript from the ScriptListener output:

1. Do one of the following:
   - Open `ScriptingListenerJS.log` on the desktop.

      At the end of the file you will see code similar to the following (although your numbers may be different):

      ```javascript
      var id19 = charIDToTypeID( "Embs" );
      var desc4 = new ActionDescriptor();
      var id20 = charIDToTypeID( "Angl" );
      desc4.putInteger( id20, 135 );
      var id21 = charIDToTypeID( "Hght" );
      desc4.putInteger( id21, 3 );
      var id22 = charIDToTypeID( "Amnt" );
      desc4.putInteger( id22, 100 );
      executeAction( id19, desc4 ,DialogModes.NO);
      ```

      **NOTE:** ScriptListener separates logged commands with horizontal lines composed of equal signs (========...). If this is not the first action recorded in the log, you can easily locate the most recent action; it follows the final equal sign line.

2. Copy the JavaScript code associated with the emboss action from `ScriptingListenerJS.log` to another file, called `emboss.jsx`.

3. In the `emboss.jsx` script, identify the values that you used with the filter (135, 3 and 100). Substitute the filter specification values with variable names.

   In the following example, 135 has been replaced with `angle`; 3 has been replaced with `height`; 100 has been replaced with `amount`.

   ```javascript
   var id19 = charIDToTypeID( "Embs" );
   var desc4 = new ActionDescriptor();
   var id20 = charIDToTypeID( "Angl" );
   desc4.putInteger( id20, angle );
   var id21 = charIDToTypeID( "Hght" );
   desc4.putInteger( id21, height );
   var id22 = charIDToTypeID( "Amnt" );
   desc7.putInteger( id22, amount );
   executeAction( id19, desc4 ,DialogModes.NO );
   ```
4. Wrap the code in a JavaScript function. In the following example, the function name is `emboss`.

```javascript
function emboss( angle, height, amount )
{
  var id19 = charIDToTypeID( "Embs" );
  var desc4 = new ActionDescriptor();
  var id20 = charIDToTypeID( "Angl" );
  desc4.putInteger( id20, angle );
  var id21 = charIDToTypeID( "Hght" );
  desc4.putInteger( id21, height );
  var id22 = charIDToTypeID( "Amnt" );
  desc7.putInteger( id22, amount );
  executeAction( id19, desc4 ,DialogModes.NO);
}
```

5. To use a JavaScript to apply the Emboss filter to a document, include the emboss function in the JavaScript and call the function with the desired parameters. For example, the following example applies the Emboss filter with angle 75, height 2, and amount 89. (See “Opening a Document” on page 28, for help in writing the code to open a document within the script.)

```javascript
// Open the document in the script
var fileRef = new File("/c/myfile")
var docRef = app.open(fileRef)

//Call emboss with desired parameters
emboss( 75, 2, 89 );
//finish the script

//include the function in the script file
function emboss(angle, height, amount )
{
  var id32 = charIDToTypeID( "Embs" );
  var desc7 = new ActionDescriptor();
  var id33 = charIDToTypeID( "Angl" );
  desc7.putInteger( id33, angle );
  var id34 = charIDToTypeID( "Hght" );
  desc7.putInteger( id34, height );
  var id35 = charIDToTypeID( "Amnt" );
  desc7.putInteger( id35, amount );
  executeAction( id32, desc7,DialogModes.NO );
}
```

6. Open Photoshop, to apply the emboss filter by selecting File > Scripts > Browse, and then browsing to the location of your emboss.jsx script. Select Open to run the script.

**Using the Action Manager from a VBS Script**

The section demonstrates how to use the contents of the `ScriptingListenerVB.log` log to create your script. Before you begin this section, you need to have already recorded an action. The example in this section assumes you have followed the instructions in “Recording a Script using ScriptListener” on page 74.

The procedures in this section use the Action Manager to make the Emboss filter available to the scripting interface.
To create a VBScript from the ScriptListener output:

1. Open ScriptingListenerVB.log from the desktop.

   At the end of the file you will see code similar to the following (although your numbers may be different):

   ```vbs
   DIM objApp
   SET objApp = CreateObject("Photoshop.Application")
   REM Use dialog mode 3 for show no dialogs
   DIM dialogMode
   dialogMode = 3
   DIM id9
   id9 = objApp.CharIDToTypeID( "Embs" )
   DIM desc4
   SET desc4 = CreateObject( "Photoshop.ActionDescriptor" )
   DIM id10
   id10 = objApp.CharIDToTypeID( "Angl" )
   Call desc4.PutInteger( id10, 135 )
   DIM id11
   id11 = objApp.CharIDToTypeID( "Hght" )
   Call desc4.PutInteger( id11, 3 )
   DIM id12
   id12 = objApp.CharIDToTypeID( "Amnt" )
   Call desc4.PutInteger( id12, 100 )
   Call objApp.ExecuteAction( id9, desc4, dialogMode )
   ```

   **NOTE:** ScriptListener separates logged commands with horizontal lines composed of equal signs (====...). If this is not the first action recorded in the log, you can easily locate the most recent action; it follows the final equal sign line.

2. Copy the VBScript code associated with the emboss action from ScriptListenerVB.log to another file, called emboss.vbs.

3. In the emboss.vbs script, identify the values that you used with the filter (135, 3, and 100). Substitute the filter specification values with variable names.

   In the following example, 135 has been replaced with angle, 3 has been replaced with height, and 100 has been replaced with amount.

   ```vbs
   DIM objApp
   SET objApp = CreateObject("Photoshop.Application")
   REM Use dialog mode 3 for show no dialogs
   DIM dialogMode
   dialogMode = 3
   DIM id9
   id9 = objApp.CharIDToTypeID( "Embs" )
   DIM desc4
   SET desc4 = CreateObject( "Photoshop.ActionDescriptor" )
   DIM id10
   id10 = objApp.CharIDToTypeID( "Angl" )
   Call desc4.PutInteger( id10, angle )
   DIM id11
   id11 = objApp.CharIDToTypeID( "Hght" )
   Call desc4.PutInteger( id11, height )
   DIM id12
   id12 = objApp.CharIDToTypeID( "Amnt" )
   Call desc4.PutInteger( id12, amount )
   Call objApp.ExecuteAction( id9, desc4, dialogMode )
   ```
4. Wrap the code in a VBScript function. In the following example, the function name is `Emboss`. The creation of the Photoshop application object needs to be outside of the function, as we will explain in the next step.

```vbnet
DIM objApp
SET objApp = CreateObject("Photoshop.Application")

Function Emboss( angle, height, amount )
REM Use dialog mode 3 for show no dialogs
DIM dialogMode
dialogMode = 3
DIM id9
id9 = objApp.CharIDToTypeID( "Embs" )
DIM desc4
SET desc4 = CreateObject( "Photoshop.ActionDescriptor" )
DIM id10
id10 = objApp.CharIDToTypeID( "Angl" )
Call desc4.PutInteger( id10, angle )
DIM id11
id11 = objApp.CharIDToTypeID( "Hght" )
Call desc4.PutInteger( id11, height )
DIM id12
id12 = objApp.CharIDToTypeID( "Amnt" )
Call desc4.PutInteger( id12, amount )
Call objApp.ExecuteAction( id9, desc4, dialogMode )
End Function
```

5. To use a VBScript to apply the Emboss filter to a document, include the `Emboss` function in the script and call the function with the desired parameters. For example, the following example applies the Emboss filter with angle 75, height 2, and amount 89. Before the script calls the function, it needs to have an open document. (See “Opening a Document” on page 28, for help in writing the code to open a document within the script.) Since the script is opening a document, it needs access to the Photoshop DOM when it calls the `Application.Open` method, so the script must create the `Photoshop.Application` object before it opens a new document.

```vbnet
DIM objApp
SET objApp = CreateObject("Photoshop.Application")

'Open the document in the script
filename = "C:\MyFile"
DIM docRef
SET docRef = objApp.Open(filename)

'Call emboss with desired parameters
Call Emboss( 75, 2, 89 )
```
Function Emboss( angle, height, amount )
REM Use dialog mode 3 for show no dialogs
DIM dialogMode
dialogMode = 3
DIM id9
id9 = objApp.CharIDToTypeID( "Embs" )
DIM desc4
SET desc4 = CreateObject( "Photoshop.ActionDescriptor" )
DIM id10
id10 = objApp.CharIDToTypeID( "Angl" )
Call desc4.PutInteger( id10, angle )
DIM id11
id11 = objApp.CharIDToTypeID( "Hght" )
Call desc4.PutInteger( id11, height )
DIM id12
id12 = objApp.CharIDToTypeID( "Amnt" )
Call desc4.PutInteger( id12, amount )
Call objApp.ExecuteAction( id9, desc4, dialogMode )
End Function

6. Apply the emboss filter script by double clicking on the file emboss.vbs. This launches Photoshop, opens the file and applies the emboss filter to the file.

Running JavaScript-based Action Manager code from VBScript

You can also access JavaScript-based Action Manager code from a VBScript using the DoJavaScriptFile method. Use the VBscript object browser for more information on the Application.DoJavaScriptFile method.

To execute JavaScript-based Action Manager code from a VBScript:

1. Follow steps 1-4 in "Using the Action Manager from JavaScript" on page 75. You will end up with a file (emboss.jsx) containing the following JavaScript code:

```javascript
function emboss( angle, height, amount )
{
    var id32 = charIDToTypeID( "Embs" );
    var desc7 = new ActionDescriptor();
    var id33 = charIDToTypeID( "Angl" );
    desc7.putInteger( id33, angle );
    var id34 = charIDToTypeID( "Hght" );
    desc7.putInteger( id34, height );
    var id35 = charIDToTypeID( "Amnt" );
    desc7.putInteger( id35, amount );
    executeAction( id32, desc7 );
}
```

2. At the end of the file emboss.jsx, add the following line of JavaScript code, which executes the emboss function with arguments passed to it from an external invocation. See Introduction to Scripting for more information about passing arguments from a VBScript to a JavaScript.

```javascript
// Call emboss with values provided in the "arguments" collection
emboss( arguments[0], arguments[1], arguments[2] );
```
3. From a VBScript you can then run the Emboss filter by saying (this example assumes `emboss.jsx` is found in C:\):

```vbnet
Set objApp = CreateObject("Photoshop.Application")

' Open the document in the script
filename = "C:\MyFile"
DIM docRef
SET docRef = objApp.Open(filename)

objApp.DoJavaScriptFile "C:\emboss.jsx", Array(75, 2, 89)
```

Running JavaScript-based Action Manager code from AppleScript

There is no Action Manager functionality in AppleScript. However, you can execute JavaScript code and files from AppleScript using the `do javascript` command. For further information, please refer to `Introduction to Scripting`.

1. Follow steps 1-4 in "Using the Action Manager from JavaScript" on page 75. You will end up with a file (emboss.jsx) containing the following JavaScript code:

```javascript
function emboss( angle, height, amount )
{
    var id32 = charIDToTypeID( "Embs" );
    var desc7 = new ActionDescriptor();
    var id33 = charIDToTypeID( "Angl" );
    desc7.putInteger( id33, angle );
    var id34 = charIDToTypeID( "Hght" );
    desc7.putInteger( id34, height );
    var id35 = charIDToTypeID( "Amnt" );
    desc7.putInteger( id35, amount );
    executeAction( id32, desc7 );
}
```

2. At the end of the file `emboss.jsx`, add the following line of JavaScript code, which executes the `emboss` function with arguments passed to it from an external invocation. See `Introduction to Scripting` for more information about passing arguments from a AppleScript to a JavaScript.

```javascript
// Call emboss with values provided in the "arguments" collection
emboss( arguments[0], arguments[1], arguments[2] );
```

3. The following AppleScript code sample opens a document and runs the Emboss filter on it:

```applescript
tell application "Adobe Photoshop CS6"
    set theFile to alias "Application:Documents:MyFile"
    open theFile
    do javascript (file <path to Emboss.jsx>) ¬
        with arguments { 75, 2, 89 }
end tell
```
Using ScriptListener to find event IDs and class IDs

The section demonstrates how to use ScriptListener to determine event IDs and class IDs for actions taken by Photoshop. These event and class IDs are used to set up notification using the Notifier class.

You can determine the event ID for any recordable action by using ScriptListener. Simply install the ScriptListener plug in, as described in “Installing ScriptListener” on page 73. Then execute the action you want to find the event ID for. The event is logged in the Script Listener log file. (See “The ScriptListener Plug-In” on page 73) If the event applies to several different classes of objects, the class ID is also logged in the log file.

The following examples show how to find the event ID for the “Open Document” event, and the event and class IDs for the “New” event, which applies to several different classes.

Finding the event ID for the “Open Document” event

1. Make sure that the ScriptListener plug in is installed.
2. Open Photoshop, then open a document.
3. Find the ScriptListener log file and open it. You can use either the VBScript log file or the JavaScript log file. In the JavaScript version of the file, you will see code that looks something like this at the end of the file, everything below the row of equal signs the log of the last action taken:

   // =======================================================
   var id14 = charIDToTypeID( "Opn " );
   var desc5 = new ActionDescriptor();
   var id15 = charIDToTypeID( "null" );
   desc5.putPath( id15, new File( "C:\Program Files\Adobe\Adobe Photoshop CS6\Samples\Fish.psd" ) );
   executeAction( id14, desc5, DialogModes.NO );

4. The executeAction method runs the action from a script, and it needs the event ID to identify which action to take. The first argument, in this case id14, provides the event ID to the method. You can see the variable id14 defined several lines earlier, and it shows that the event ID for the Open Document action is "Opn ".

5. You can now use this event ID to set up event notification on Open Document from your scripts. In JavaScript, for example:

   var eventFile = new File(app.path + "/Presets/Scripts/Event Scripts Only/Welcome.jsx")
   app.notifiers.add("Opn ", eventFile)

Finding the event ID and class ID for the “New” event

1. Make sure that the ScriptListener plug in is installed.
2. Open Photoshop, then create a new document using File > New.
3. Next, create a new channel, using the Create New Channel icon on the Channels palette.
4. Find the ScriptListener log file and open it. You can use either the VBScript log file or the JavaScript log file. We have recorded two actions, so we are interested in looking at the last two sections in the file that are delimited by the rows of equal signs. In the JavaScript log file, you will see code that looks something like this:
CHAPTER 4: Action Manager

Using ScriptListener to find event IDs and class IDs

5. The first section represents the scripting code to execute the “New Document” event. The second section represents the scripting code for the “New Channel” event.
6. The `executeAction` method for both of these actions takes an argument whose value is defined as "Mk ". (See id17 and id34.) This is the event ID for the “New” action. This action also needs to know what class to use, the class ID for the event.

7. The `putObject` method identifies the class the action operates on. The second argument to `putObject` provides us with the class ID that we need. In the first action, `id33` is defined as "Dcmn", in the second action, `id45` is defined as "Chnl". These provide our class IDs for Document and Channel, respectively.

8. You can now use these event and class IDs to set up event notification on the New Document and New Channel events from your scripts. In JavaScript, for example:

```javascript
var eventFile = new File(app.path +
    "/Presets/Scripts/Event Scripts Only/Welcome.jsx")
app.notifiers.add("Mk ", eventFile, "Dcmn")
app.notifiers.add("Mk ", eventFile, "Chnl")
```
Index

A

Action Manager
defined, 73
running JavaScript code from AppleScript, 80
running JavaScript code from VBScript, 79
scripting objects, 74
using from JavaScript, 75
using from VBScript, 76
actions
vs. scripts, 8
working with, 73
Actions palette, 73
active objects, setting, 25
Adobe Photoshop object model, 11, 33
AppleScript
conventions, 6
creating, 18
executing JavaScript from, 10
running, 18
unit value considerations, 56
Applescript
viewing dictionary, 21
Application object
defined, 12
display dialogs, 33
referencing, 22
relationship to user interface, 14
targeting, 22
using, 34
Art Layer object
adding in JavaScript, 25
adding in VBScript, 24
applying styles, 40
creating, 37
defined, 12
filters, 53
making text layer, 41
referencing, 38
relationship to user interface, 15
working with, 36

C
calculations, unit values, 57
Channel object
activating, 28
changing type, 46
defined, 13
relationship to user interface, 15
setting the active channel, 28
working with, 46
Channel object, kinds of, 13
class IDs, finding with ScriptListener, 81
classes, finding, 21
clipboard commands, 54
collections, VBScript indexing, 11
Color object
in the DOM, 16
Color objects
applying to text, 64
comparing, 53
defined, 51
getting and converting, 52
setting hex values, 52
solid color classes, 52
web safe, 53
working with, 51
Color Sampler object
defined, 14
relationship to user interface, 15
commands
conventions, 6
viewing, 21
component channels, 13
conditional logic, 8
constants
defined, 16
finding, 16, 21
containment hierarchy, 11
conventions, 6
copy and paste commands, 54
copy merged command, 55
Count Item object
defined, 14
relationship to user interface, 15

D
dialogs, controlling, 33
display dialogs, 33
Document Info object
defined, 13
relationship to user interface, 15
using, 47
Document object
  activating, 26
  adding, 24
  defined, 12
  document information, 47
  manipulating, 35
  opening, 28
  relationship to user interface, 14
  saving, 31
  unit values, 58
  using, 34
document object model (DOM), See object model

E
  enumerated values
    finding, 21
  EPS open options object, unit values, 58
  event IDs, finding with ScriptListener, 81
  event notification, setting up, 48

F
  file extensions
    script files, 9
  files
    inferring format, 28
    opening, 28
    opening using specific settings, 29
    saving, 31
    specifying format, 29
  filters
    additional, 54
    applying motionblur, 71
    applying wave, 66–70
    making scriptable, 74
    working with, 53

H
  Hello World script, 17–20
  hex color values, setting, 52
  hierarchy, 11
History State object
  defined, 13
  purging, 48
  relationship to user interface, 15
  reverting, 48
  using, 47
history states
  defined, 47

I
  images, changing composition, 12

J
  JavaScript
    conventions, 6
    creating, 19
    executing, 10
    executing from AppleScript, 10
    executing from VBScript, 10
    running, 19
    support, 10
    using Action Manager, 75
    workflow automation sample, 60

L
  layer classes, 12
Layer Comp object
  defined, 13
  relationship to user interface, 15
Layer objects
  activating, 27
  adding, 25
  applying styles, 40
  creating, 37
  defined, 12
  determining kind, 41
  linking, 40
  referencing, 38
  testing for text layers, 41
  working with, 36
Layer Set object
  creating, 38
  defined, 12
  relationship to user interface, 15
  working with, 36, 39
lens flare open options object, unit values, 58
logic, conditional, 8

M
  masked area channels, 13
Measurement Scale object
  defined, 14
  relationship to user interface, 15
measurement units
  document preferences, 61
  working with, 56
merged copies, 55
metadata defined, 15
methods
  conventions, 6
  viewing, 21
motionblur filters, defining, 71

N
Notifier object
  defined, 14
  finding class IDs, 81
  finding event IDs, 81
  relationship to user interface, 15
  using, 48

O
object model
  concepts, 11
  working with, 33
objects
  Also see individual objects
  activating, 25
  Adobe Photoshop object model, 11
  creating in a script, 23–25
  hierarchy, 11
  viewing, 21
offset filter object, unit values, 58
Open options classes, 16

P
parent objects defined, 26
paste commands, 54
Path Item object
  creating a straight line, 49
  defined, 14
  relationship to user interface, 15
Path Point object defined, 14
paths, creating, 49
PDF open options object, unit values, 58
Photoshop object model, 11
preferences
  setting, 32
  working with, 61
Preferences object
  defined, 14
  relationship to user interface, 15
properties
  conventions, 6
  finding, 21

R
ruler units
  defined, 56
  setting, 59
  value usage, 57
  values, 56

S
Save options classes, 16
saving documents, 31
Script Editor
  using, 18
scripting languages
  example scripts, 17
  supported, 9
ScriptListener
  finding class IDs, 81
  finding event IDs, 81
  installing, 73
  log files, 73
  recording scripts, 74
  uninstalling, 74
scripts
  advanced, 60
  capabilities, 8
  creating, 60
  creating objects, 23–25
  defined, 8
  executing, 10
  file locations, 10
  functionality, 9
  recording, 74
  startup, 10
  valid file extensions, 9
  vs. actions, 8
selected area channels, 13
Selection object
  creating, 43
  defined, 13
  defining area, 67
  feathering, 44
  filling, 45
  inverting, 44
  loading, 45
  relationship to user interface, 14
  resizing, 44
  restoring, 46
  storing, 45
  stroking, 44
  working with, 42
Solid Color classes, 52
spot color channels, 13
startup scripts, 10
stroking
  selections, 44
  text, 64
styles, applying to layers, 40
Sub Path Item object defined, 14

T
text
  applying color, 64
  formatting, 42
  layers, 41
  stroking, 64
Text Item object
  creating, 41
  defined, 12
  formatting text, 42
  unit values, 58
  working with, 41
text layers, 41
The, 54
type library, VBScript, 22
type units
  defined, 56
  setting, 59
typographic conventions, 6

U
units
  AppleScript considerations, 56
  as parameters, 58
  as properties, 57
  in arguments, 58
  in calculations, 57
  setting, 59
  special types, 56
  value usage, 57
  values, 56
  working with, 56

V
value types
  constants, 16
VBScript
  conventions, 6
  creating, 19
  executing JavaScript from, 10
  running, 19
  type library, 22

using Action Manager, 76

W
wave filters, applying, 66–70
web safe color, 53
workflow automation, JavaScript, 60