Photoshop allows two kinds of macros: actions and scripts. Scripts are higher level than actions. Scripts let you edit nitty-gritty details and also let you do very general things. With a script you could say “Go through the folder ‘IMAGES’ and open every file that has ‘_P_’ or ‘Profile’ in its name, and perform action ‘color_correct’ on those images.’ A script also allows you to write what programmers refer to as “conditional logic” which means you can have statements that in English would start with the word “if” and be followed by a phrase that in English would start with the word “then”. An example would be: “Find every file that ends with ‘.eps’ and open it. If it contains a path named ‘P56’ then set the foreground color to RGB 255, 0, 0 and stroke the path with 10 pixel brush.”

This next bit is from the built-in Photoshop manual, on everyone’s hard drive:

>>>>>>>>>>>>>>>>>>
1.3 Why use scripting? Graphic design is a field characterized by creativity, but aspects of the actual work of illustration and image manipulation are anything but creative. Scripting provides a tool to help save time spent on repetitive production tasks such as resizing or reformatting documents. Start with short, simple scripts such as those described in this manual and move on to more involved scripts as you become more proficient. Any repetitive task is a good candidate for a script. Once you can identify the steps and conditions involved in performing the task, you’re ready to write a script to take care of it.

1.4 What about actions? Photoshop actions are different from scripts. A Photoshop action is a series of tasks you have recorded while using the application—menu choices, tool choices, selection, and other commands. When you “play” an action, Photoshop performs all of the recorded commands. Actions and scripts are two ways of automating repetitive tasks, but they work very differently.
• You cannot add conditional logic to an action. Unlike a script, actions cannot make decisions based on the current situation.
• A single script can target multiple applications. Actions can’t. For example, you could target both Photoshop and Illustrator in the same script. The Actions palette, invoked under the Window menu, supports actions with a great deal of sophistication (including the ability to display dialogs) and allows users to work with selected objects, as illustrated below.
>>>>>>>>>>>>>>>>>>>

So a script is a bunch of words that you type into a blank text document, and then save to your hard drive. If you put the file in a special Photoshop folder, then when Photoshop is running the script will be listed under the File>Scripts menu. Which folder do you want to save your file to? On a Windows machine, you'd look in My Computer, then go into the C drive, then go into the Program Files folder, then into the Adobe folder, then into the Photoshop folder, then into the Presets folder, then into the Scripts folder.

Scripting, even more than actions, involves programming, so to do scripting you have to understand what a “variable” is. A variable is a word that you can use as a container.

Think of it as a box that you can put stuff in and take stuff out of. You can use almost word you want for the variable. You create a variable like this:

```
var howIFeelAboutTheWorld;
```

Now you have a variable called “howIFeelAboutTheWorld”. Its starts off with no value. You could give a value if you wanted to:

```
howIFeelAboutTheWorld = “The world is basically a good place.”;
```

Now you’ve stored text inside of it. It’s like a box and you’re placing words in it. If you want to see what’s in the box, use the special word ‘alert’ which is a built-in command:

```
alert(howIFeelAboutTheWorld);
```

A dialog box would open and it would have the words “The world is basically a good place.”

A variable is called a variable because the stuff inside of it can vary. You can change what is inside of howIFeelAboutTheWorld:

```
howIFeelAboutTheWorld = “The world is a bad place and I don’t like it.”;
```

If you now do this:

```
alert(howIFeelAboutTheWorld);
```

Then the words “The world is a bad place and I don’t like it.” would appear on screen in a dialog box.

To do useful stuff in Photoshop, we have to use more of the special, built-in commands. Two of them are “documents” and “length”. Suppose you currently have 5 images open in Photoshop and you run a script that has these three lines:

```
var howManyWindowsAreOpen;
howManyWindowsAreOpen = documents.length;
alert(documents.length);
```

A dialog box would appear with the number “5” in it, telling you that you’ve currently 5 images open.

Not to complicate things, but Photoshop speaks 3 languages, kind of like a person who knows German, French and English. Photoshop knows one language that only works on Macintosh computers (Applescript) and it knows another language that only works on Windows (Visual Basic or VBscript) and it knows a 3rd language that works on both (Javascript). To my mind, one should always write scripts in Javascript and that way the script will work on both Windows and Macintoshes. From here on out, I’m only going to talk about Javascript.

This next bit is from the Photoshop built-in manual:

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The singular advantage to writing JavaScripts is that the scripts run on any platform, regardless of the underlying hardware or operating system. JavaScript is consequently the language of choice for developers not wishing to lock themselves into a single proprietary platform. Use the text editor of your choice to create JavaScripts.

To make scripts accessible to Photoshop, drag-and-drop your files directly into the "Presets>Scripts" folder.

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

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

A collection of JavaScripts comes pre-installed with Photoshop. These scripts display alongside the Scripts menu, as illustrated above. You can use these scripts “out-of-the-box” or create your own. Use the “Browse” option to locate scripts you’ve created in other directories or to find scripts that reside on a network. On both Mac and Windows, a JavaScript file must be saved as a text file with a '.js' file name extension.

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

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

But how do you do really cool stuff with a script, like create new layers, write text to the layer, color the text, or run different filters based on what’s in the file? The first thing you have to do is create a variable that refers to the current document (when I say “refer” imagine its like a pointer that tells the script which image you’re talking about). First create the variable and give it name that makes sense:

```
var docRef;
```

Then you use more special commands to create the reference:

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

Now you can use docRef to refer to any aspect of the currently active document.

Once we have docRef, we can add a new layer to the current document like this:

```
docRef.artLayers.add();
```

We can create a reference to the new layer like this:

```
newTextLayer = docRef.artLayers.add();
```

Now we can use newTextLayer to refer to any aspect of the new layer.

Suppose we wanted to write red text to the new layer. First we define what color of red we want. We will use RGB 8-bit color, which means a number from 0 to 255 for red and then another number from 0 to 255 for green, and then another number from 0 to 255 for blue. All combined, this allows for 16 million different colors. When we go like this:

```
var textColor;
textColor = new SolidColor;
textColor.rgb
```

We are letting Photoshop know that we are creating an RGB color. If we instead went like this:

```
textColor.cmyk
```

then Photoshop would know we were creating a CMYK color. “SolidColor” is a built-in type for Photoshop and when we do this:

```
textColor = new SolidColor;
```

what we are saying is “From now on, the variable textColor is a specific instance of the overall type known as SolidColor.”

So to create a new RGB color, and give it a pure red, we go like this:

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

Next, lets create a variable to hold our text, and then put some text in it:

```
var theCityILiveIn;
theCityILiveIn = "I live in Richmond.”;
```

Now we need to take the layer we created and make a text layer. Layers have a property called “kind” that determines what kind of layer they are. We make a layer a text layer by doing this:

```
newTextLayer.kind = LayerKind.TEXT;
```

Next we give the layer some content:

```
newTextLayer.textItem.contents = theCityILiveIn;
```

Then we position it, saying how far we want it from the top and from the left side, in inches. First we have to tell Photoshop that we're going to be talking using inches (instead of, say, pixels):

```
preferences.rulerUnits = Units.INCHES;
```

Then we state our position, and Photoshop knows these numbers are inches from the top and left:

```
newTextLayer.textItem.position = Array(0.75, 1);
```

Then we tell Photoshop what size the font should be:

```
newTextLayer.textItem.size = 36;
```

Then we tell Photoshsop was color the text should be:

```
newTextLayer.textItem.color = textColor;
```

So now we've created a new layer and written words to that new layer.

To do really cool stuff we have to know all the special commands at our disposal. They are listed in the Javascript Reference Guide, which is automatically installed whenever Photoshop is installed on a computer. On a Windows machine, you'd look at the C drive, then inside the folder "Program Files" and then inside the folder "Adobe" and then inside the folder "Scripting Guides". It's a rich language that lets you refer to any layer, path, path point, selection, document, folder, and it even allows you to use dialog boxes to ask for feedback from the user.

Here is an example script that comes built-in with the manual:

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

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

```
// Hello Word Script

// Remember current unit settings.
// We're going to switch to inches.
```

```
 // If currently the unit setting is pixels, we will want  to set it
// back to pixels after the script is done.

var originalUnit = preferences.rulerUnits;


// Now we're going to set the Photoshop measurement unit to inches.

preferences.rulerUnits = Units.INCHES;


// Create a new 4x4 inch document and assign it to a variable

var docRef = app.documents.add( 4, 4 );


// Create a new art layer

var artLayerRef = docRef.artLayers.add();


// Tell Photoshop that this layer should be a text layer:

artLayerRef.kind = LayerKind.TEXT;


 // Set the contents of the text layer.

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


 // Now the script is done. We want to set all the variables to "null". That way if another
// script runs, these variables start off blank,  rather than having whatever value and
// text we just gave them. You might get strange bugs if you ran several scripts and
// they used variables with the same name, and you forgot to set those variables
// to blank value between scripts.

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


// Restore original ruler unit setting. If the default was pixels instead of inches,
// we are setting it back to pixels.
```

```
app.preferences.rulerUnits = originalUnit;
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Here is another example script. It takes the foreground color and tells you how much red, green and blue is in that foreground color:

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
var fgColor = new SolidColor();
fgColor = app.foregroundColor;

var fgRGBColor = fgColor.rgb;
alert("Red:" + fgRGBColor.red);
alert("Green:" + fgRGBColor.green);
alert(" Blue:" + fgRGBColor.blue);
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

For H & S, we might want a script that looked for spelling errors in the names of the “White Gold”, “Yellow Gold” or “Diamond” channel. When there was both white gold, yellow gold, and diamonds, the actions created a path called Yellow Gold, but one had to carve the White Gold out of it and redo it. So the possibility of spelling errors exists. In this next script, when everything is spelled correctly, nothing happens, but when something is misspelled, then we use the alert() command to bring it to the attention of the person running the script.

```
// get a reference to the active document
var docRef = app.activeDocument;

// get a list of all the paths
var myPaths = docRef.pathitems;

// For now, lets make pretend that the paths are misspelled. If later we find they are
// correctly spelled, we will change this. We will create two variable to keep track of
// whether the paths are correctly spelled and we will give them a value of "false".
var yellowGold = false;
var whiteGold = false;

// if there is just one path then this image is probably a pure gold ring, so we don't need to
// check for misspellings. So we only need to run this check if the length of the list of
// paths is  greater than one.
if (1 < myPaths.length) {
```

```
// this next line initiates a loop that will allow us to go through and check
// every path.
for (var pathIndex=0; pathIndex < myPaths.length; pathIndex++) {

    // this next line gets us one item from the list.
    // Every time we go through the loop it will get the next item in
    // the list.
    var thisPath = myPaths [pathIndex];

    // now we get the name of this path
    var nameOfPath = thisPath.name;

    // Is it spelled "Yellow Gold"?
    if (nameOfPath == "Yellow Gold") {
        yellowGold = true;
    }

    Is it spelled "White Gold"?
    if (nameOfPath == "White Gold") {
        whiteGold = true;
    }
}

// Okay, if there were only two paths, and one of them was "Path 1"
// then the second one must be either "Yellow Gold" or "White Gold".
// If either are spelled correctly, then we can say that the path was
// spelled correctly.
if (myPaths.length == 2) {

    // this next line says "If yellowGold or whiteGold is true"
    if (yellowGold || whiteGold) {
        // all is good, do nothing
    } else {
        alert ("One of the paths is misspelled.");
    }
}

// If there are 3 paths, then both the white gold and yellow gold path
// must be present. So both white gold and yellow gold should
// be true.
if (myPaths.length == 3) {

    // this next line says "If both yellowGold and whiteGold are true"
    if (yellowGold && whiteGold) {
        // all is good, do nothing
```

```
                } else {
                        alert ("One of the paths is misspelled.");
                }
        }

}
```