# Extending Configurator with Scripts

Note: Programmers will often interchange the terms "function," "method," "Routine," "procedure," or "subprogram" for the same thing. For the sake of simplicity and consistency, we'll use the term "method" to describe commands/tasks that are built into the Photoshop DOM and we'll use the term "function" to describe custom commands/tasks that are outside of the Photoshop DOM.

## Launching the Extend Script Toolkit 2.

The ExtendScript Toolkit is the text editor you will use to write ExtendScript for these lessons. The ESTK helps make writing scripts easier through syntax-highlighting, auto-complete, and built in debugging. The ESTK is installed when you install Photoshop or the Creative Suite:

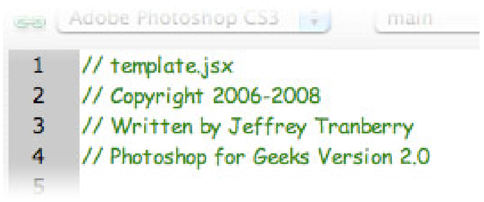
Mac: <hard drive>/Applications/Utilities/Adobe Utilities/ExtendScript Toolkit CS4/
Windows: C:\Program Files\Adobe\Adobe Utilities\ExtendScript Toolkit CS4\

## Writing your first script.

When creating scripts for Configurator, I will always create, test and save my scripts in the ESTK before moving them into Configurator. It makes coding, debugging and archiving far more easy than scripting directly in Configurator.

I rarely start writing a script from scratch. I'll usually start with a template or existing script and modify it to solve a new workflow problem. I've created several template scripts, which are available on your class CD and website, to help you get started. The most basic template is called template.jsx.

Let's look at the first 4 lines of the script. These are what programmers call "comments." Comments are notes to yourself, or others you share your scripts with, detailing information about script, and what each piece of code does. Notice that the color of the comments is green. The ExtendScript Toolkit color-codes different pieces of text so you can tell comments from code, and make the script more readable.

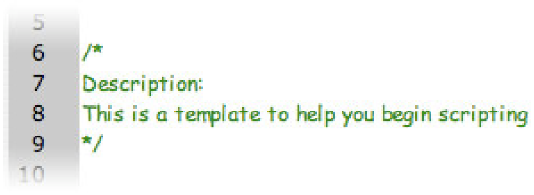


```
1   // template.jsx
2   // Copyright 2006-2008
3   // Written by Jeffrey Tranberry
4   // Photoshop for Geeks Version 2.0
5
```

You create single-line comments by beginning the line with two forward slashes // or you can create double-line comments by beginning with a forward slash and an asterisks /* and ending it will an asterisk and a forward slash */.

In the case of these four lines, I'm giving some information about the name of the script; when it was written, who wrote it, and what version of the script is if I'm modifying and releasing new versions of the script.

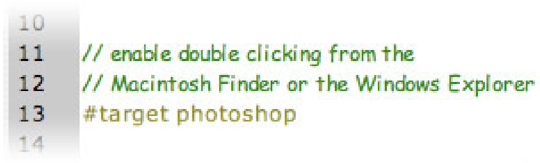
In lines 6-9, I'm using a multi-line comment to give a detailed description of what my script does.



```
5
6   /*
7   Description:
8   This is a template to help you begin scripting
9   */
10
```

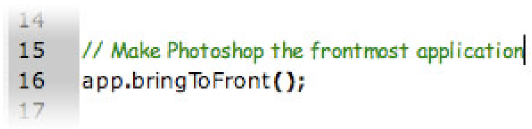
Up to this point, everything in this script has been a comment, and hopefully, this gives you a clue that comments are just as important, if not more important, as working code. This will be common theme as we work on scripts. Commenting will make working with or sharing scripts much easier.

The next chunk of code actually does something. Lines 11 and 12 describe what the code on line 13 does. In this case, #target photoshop tells the script that it's been created to run in Photoshop if it's been double-clicked in the finder or run from the ExtendScript Toolkit. Since scripts can run in other Creative Suite applications like Bridge, Illustrator, or InDesign, so it's always good form to include this code in every script you write.



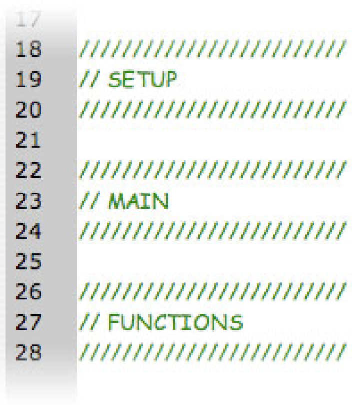
```
10
11  // enable double clicking from the
12  // Macintosh Finder or the Windows Explorer
13  #target photoshop
14
```

Line 15 is a comment for the method in line 16 `app.bringToFront();` which brings Photoshop to the front when a script is running. I like to include this method in all my scripts because I like to see my scripts do their magic. There's something satisfying about watching Photoshop run under autopilot.



```
14
15  // Make Photoshop the frontmost application
16  app.bringToFront();
17
```

The next three chunks of comments are what I call section headers. I usually organize my scripts into 3 parts: SETUP, MAIN, and FUNCTIONS.



```
////////////////////////
// SETUP
////////////////////////

////////////////////////
// MAIN
////////////////////////

////////////////////////
// FUNCTIONS
////////////////////////
```

In the SETUP section, I'll usually declare variables for my script. The MAIN section is where I'll put the working code. The FUNCTIONS section is where I'll define any custom functions that I'll call in the MAIN section of the script. We'll talk more about variables and custom functions as we start to write more complex scripts later on.

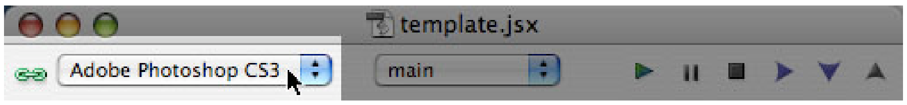
## The alert(); method.

The first method we're going to learn is the `alert();` method. An alert simply pops a dialog and displays some information. I believe alerts are an invaluable tool for learning scripting and they will also come in handy later as you start debugging more complex scripts. We'll use alerts for several of our first exercises.

For our first alert, we want to display the text "Hello, World!"

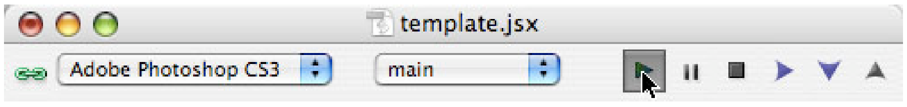
In our template.jsx script, lets first make a comment of what we want our alert to do. Then on the next line we'll type in our method `alert();` and pass in the string "Hello, World!" as an argument in between the parenthesis:

Let's test our script. Because I have the command #target photoshop in my script, I don't need to specify a target in the ESTK. If I didn't have the command #target photoshop I'd have to manually select "Photoshop" from the pop-up in the upper left-hand corner of the script window.



If the link icon next to the pop-up is broken and red, that usually means that Photoshop isn't launched and running. You can either manually launch Photoshop or run your script and the ESTK will ask if you want to launch Photoshop or not. If Photoshop is launched and your scripts fail, make sure you are targeting the right program. Often times, I'll forget to include the command for #target photoshop and the script will try to run in the ESTK instead of Photoshop.

Click the play button in the upper right-hand corner of the script window to play your script.

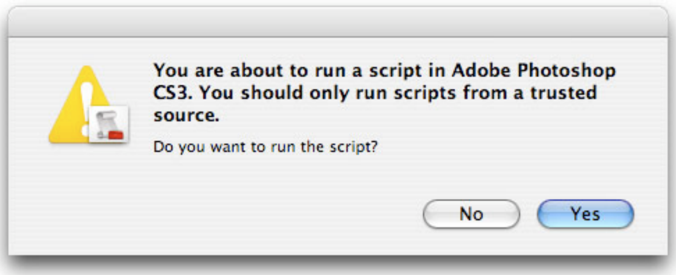


Photoshop should come to the front and display a dialog with our "Hello, World!" text in it.

Once we've tested our script and we know it works, let's finish commenting and save our script helloWorld.jsx to the desktop.

## Running and Installing Scripts.

To run a script from the finder, simply double-click the script icon or drag the script onto the Photoshop application. If you run a script this way, you will be asked whether you want to run the script or not.



**You are about to run a script in Adobe Photoshop CS3. You should only run scripts from a trusted source.**

Do you want to run the script?

To get your script to show up in Photoshop's File>Scripts menu, navigate to the Presets>Scripts folder inside the Photoshop application folder on your hard drive and copy or place your script in the folder.

Mac: <hard drive>/Applications/Adobe Photoshop CS4/Presets/Scripts/
Windows: C:\Program Files\Adobe\Adobe Photoshop CS4\Presets\Scripts\

If Photoshop is already running, you'll need to quit and restart Photoshop for the script to show up in the menu.

When you run scripts from inside Photohop, you won't be asked whether you want the script to run or not, and the script will just execute.

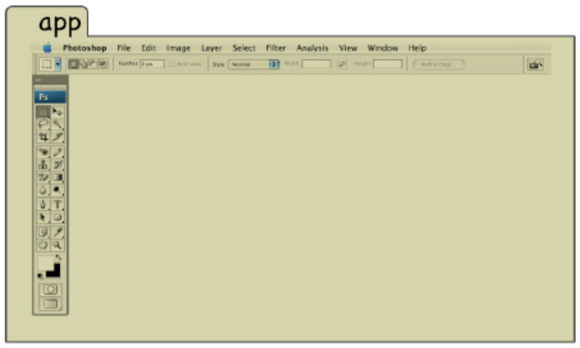
You can assign a keyboard shortcut to your script using the Edit>Edit Shortcuts… dialog. Tip: Don't use the [option] key for a keyboard shortcut for a script. The option key puts the script into "debug" mode and will launch the ESTK. You can also record an action that runs a script, and assign an F-key to the action to run it.

## The Photoshop DOM.

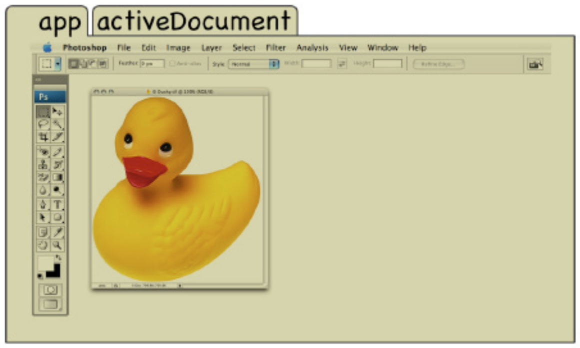
Let's start exploring the Photoshop DOM. What is a DOM? DOM stands for Document Object Model. Document Object Model is a programming interface that lets you interact with Photoshop to examine and manipulate the Photoshop application, Photoshop documents, and the objects contained within the document.

One way to think of the DOM is to think of a series of folders put inside one another. You can imagine either physical folders, or electronic folders on your hard drive or like a website path.

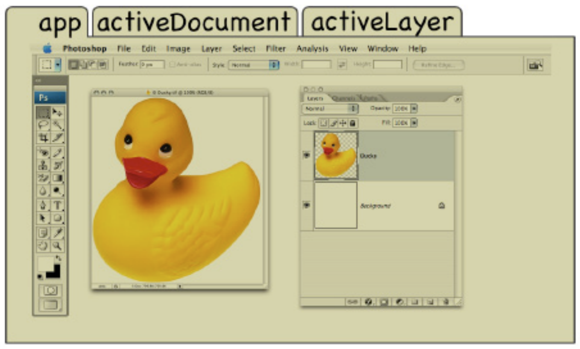
Each folder can contain various objects. Imagine the first, outside folder containing everything else in side it. That's Photoshop or `app`.



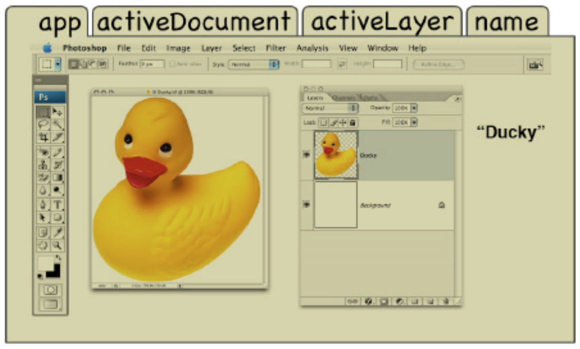
Now imagine there's another folder inside the Photoshop folder. That's the Photoshop document you are working on, or the `activeDocument`.



Now imagine there's another folder nested inside the document folder. That's the layer you are working on, or the `activeLayer`.

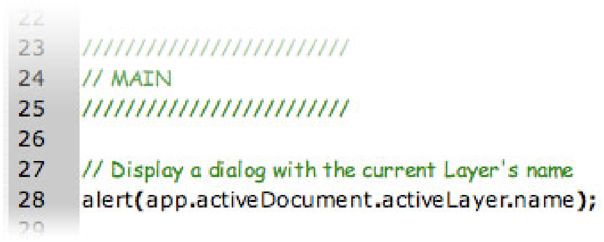


Now imagine the layer folder contains several sheets of paper in it. Each sheet of paper has a unique piece of information about the layer: it's name, it's visibility, it's opacity, it's blendmode, whether it's locked or not, etc.



So, if I wanted to look at the sheet of paper with the current layers name, I'd need to go into the Photoshop folder, then the document folder, then the layer folder, and look for the sheet with name on it. In the scripting world, the code `app.activeDocument.activeLayer.name` would be how you get the current layer's name.

Let's have Photoshop show an alert with the current layer's name. Start with the [template.jsx](template.jsx) script and add an alert, but this time, let get the current layer's name and display that in the alert dialog:

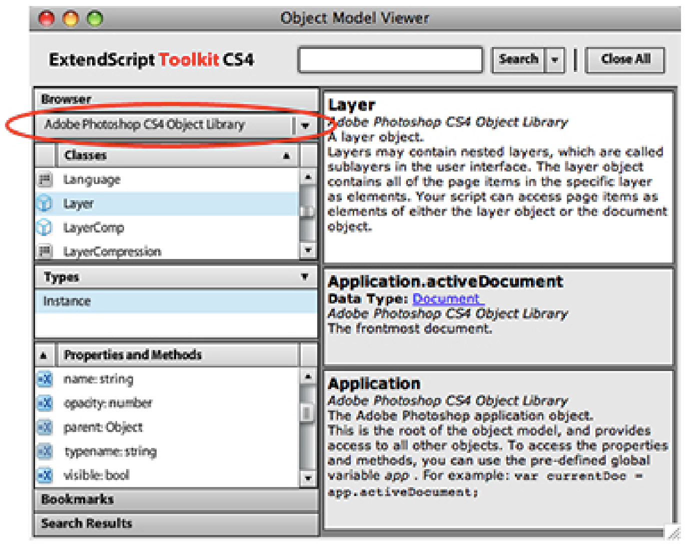


```
/////////////////////////
// MAIN
/////////////////////////

// Display a dialog with the current Layer's name
alert(app.activeDocument.activeLayer.name);
```

Make sure there is a document open in Photoshop that contains some layers and one of the layers is selected. Press play to test our script. Photoshop displays a dialog with the current layer's name on it. ([layerName.jsx](layerName.jsx))

## Using the Object Model Viewer to explore the Photoshop DOM.

Now that we know a little bit about how Photoshop Document Object Model works, let's take a look at Object Model Viewer. In the ExtendScript Toolkit, go to the Help menu and choose Object Model Viewer. Make sure the Browser accordian/pane is visible and choose "Adobe Photoshop CS4 Object Library" from the classes pop-up.



**Layer**
*Adobe Photoshop CS4 Object Library*
A layer object.
Layers may contain nested layers, which are called sublayers in the user interface. The layer object contains all of the page items in the specific layer as elements. Your script can access page items as elements of either the layer object or the document object.

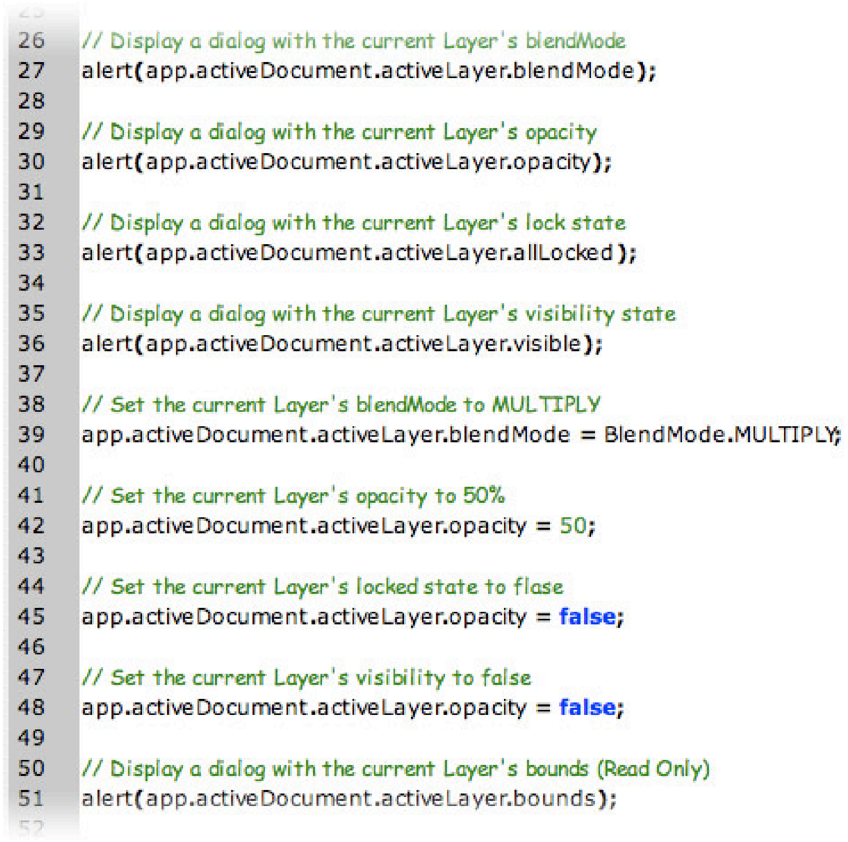
**Application.activeDocument**
**Data Type:** Document
*Adobe Photoshop CS4 Object Library*
The frontmost document.

**Application**
*Adobe Photoshop CS4 Object Library*
The Adobe Photoshop application object.
This is the root of the object model, and provides access to all other objects. To access the properties and methods, you can use the pre-defined global variable *app* . For example: `var currentDoc = app.activeDocument;`

Let's navigate through the object classes and inspect the properties and methods associated with each class. Click on Application in the classes panel of the Object Model Viewer to examine all the properties and methods of the `Application` or `app`. Second, select `activeDocument` from the properties and methods panel. We can now dig deeper into the DOM, by selecting `Layer` in the classes panel.

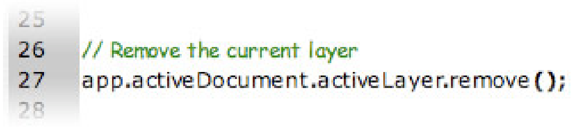
Notice how we can see all the properties and methods of the `activeLayer` in the properties and methods panel? Properties are preceded by a blue icon and methods are preceded by a red icon. Methods are always followed by `()`.

We can see the property for name that we used in the previous example. Experiment alerting and setting the value of some other properties such as `blendMode`, `opacity`, `allLocked`, and `visibile`. Note that some properties like `bounds`, `typename`, and `parent` are "Read Only" meaning that you can get and alert their value, but you can't set their value. ([layerProperties.jsx](layerProperties.jsx))



```
// Display a dialog with the current Layer's blendMode
alert(app.activeDocument.activeLayer.blendMode);

// Display a dialog with the current Layer's opacity
alert(app.activeDocument.activeLayer.opacity);

// Display a dialog with the current Layer's lock state
alert(app.activeDocument.activeLayer.allLocked);

// Display a dialog with the current Layer's visibility state
alert(app.activeDocument.activeLayer.visible);

// Set the current Layer's blendMode to MULTIPLY
app.activeDocument.activeLayer.blendMode = BlendMode.MULTIPLY;

// Set the current Layer's opacity to 50%
app.activeDocument.activeLayer.opacity = 50;

// Set the current Layer's locked state to flase
app.activeDocument.activeLayer.opacity = false;

// Set the current Layer's visibility to false
app.activeDocument.activeLayer.opacity = false;

// Display a dialog with the current Layer's bounds (Read Only)
alert(app.activeDocument.activeLayer.bounds);
```

Let's try working with some methods. We can `remove();` the layer, which does the same thing as dragging the layer to the trash button in the layers palette.

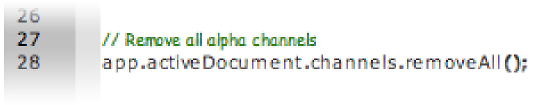


```
// Remove the current layer
app.activeDocument.activeLayer.remove();
```

Create a new document with a few layers and try running a script that uses the `remove();` method. (deleteLayer.jsx)

### Delete All Channels.

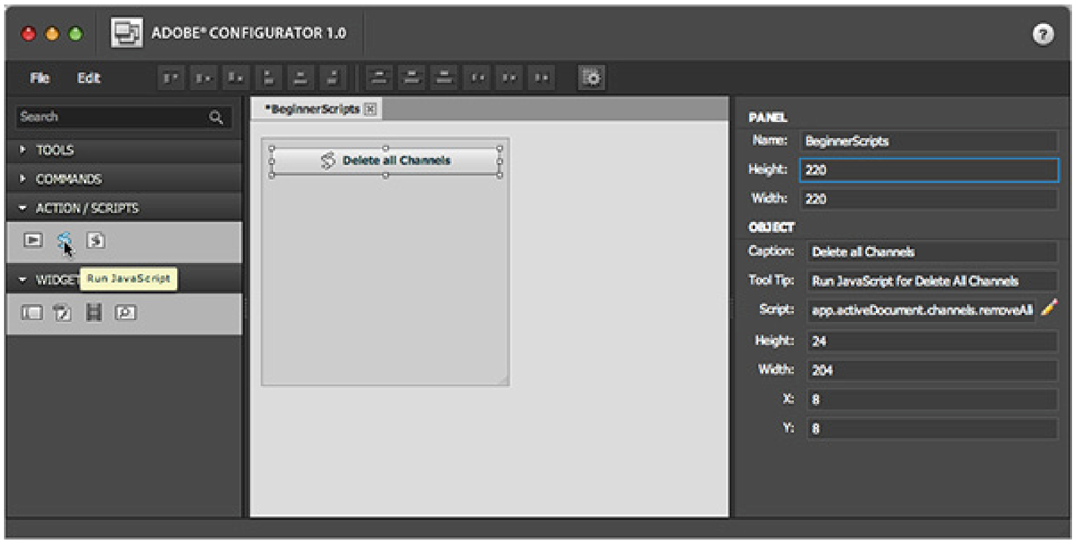
Let's look at a method that's not exposed as a menu command in Photoshop. We want to remove all the alpha channels in the document. In the Classes panel, scroll down to Channels and look at its properties and methods in the properties and methods panel. Notice there is a method for `removeAll()`. Let's try the code
(1_removeAllAlphaChannels.jsx)



```
// Remove all alpha channels
app.activeDocument.channels.removeAll();
```

### Creating a new Panel and Run JavaScript button

In Configurator choose File>New Panel. Name the panel in the righthand inspector panel. Next drag a Run JavaScript object onto the panel canvas from the ACTION/SCRIPTS panel. With the Run JavaScript object selected, add a Caption for the name of the button and a Tool Tip description of the the button will do in the righthand inspector panel. Next, copy just the code from (1_removeAllAlphaChannels.jsx) and paste it into the Script field.



PANEL
Name: BeginnerScripts
Height: 220
Width: 220

OBJECT
Caption: Delete all Channels
Tool Tip: Run JavaScript for Delete All Channels
Script: app.activeDocument.channels.removeAll
Height: 24
Width: 204
X: 8
Y: 8

Tips: 1) Configurator doesn't maintain the formatting of your script. Always save a .jsx file to retain the formatting of your script for your reference. 2) Because Configurator doesn't maintain the formatting, don't include comments when you copy your code from your script. Comments in your code will break your scripts in Configurator.

Choose File>Export Panel to export the panel. Configurator will automatically target the Plug-ins/Panels folder. Quit/restart Photoshop to load the panel. Open the panel from the Window>Extensions menu. Create or open a document with with multiple alpha channels and press the [Delete all Channels] button to test your script.

## Improving the user experience.

Open the history panel. Notice how a history step was created for each channel that was removed. If a user does an undo, it won't undo all the history steps. We can control this with scripting and the `suspendHistory()` method. (2_removeAllChannelswithSuspendHistoryCommented.jsx)

```
// create a variable for the current doc
var doc = app.activeDocument;

/////////////////////////
// MAIN
/////////////////////////

// suspendHistory method. Creates one history step named "Delete All Channels" from all steps inside the function named main();
doc.suspendHistory("Delete All Channels", "main()");

/////////////////////////
// FUNCTIONS
/////////////////////////

// a container function for all the steps we want to collapse down into one history step using suspendHistory
function main(){
    // Remove all Alpha Channels
    doc.channels.removeAll();
}
```

## Create a multi-step workflow for production

Let's add some more steps to create a multi-step workflow. We'll flatten the document then remove alpha channels and paths.

```
// a container function for all the steps we want to collapse down into one history step using suspendHistory
function main(){
    // Flatten the layers if any
    doc.flatten();
    // Remove all alpha channels
    doc.channels.removeAll();
    // Remove all paths
    doc.pathItems.removeAll();
}
```

Create two more buttons in Configurator using Run JavaScript objects and copy/paste your code into the script field. Export the panel and test the results.

## Closing a document.

After we save our document, we'll want to close it and work on another document.

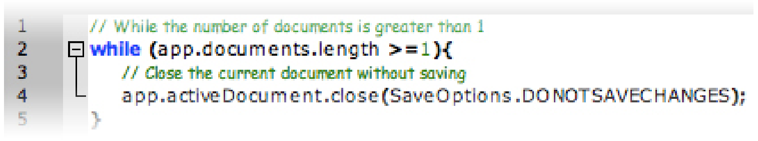
To do this, we'll use the `close();` method on the `activeDocument`:

```

// Close without saving
app.activeDocument.close(SaveOptions.DONOTSAVECHANGES);

```

You can specify whether to `DONOTSAVECHANGES`, which closes the document without saving any changes. This is the option I use most often since I'm generally closing documents I know I just saved. You can also specify to `SAVECHANGES`, which saves and closes the document. The last option is `PROMPTTOSAVECHANGES`, which displays a dialog asking the user whether they want to save their document or not. Since I usually want our scripts to run without user interaction, I rarely use this option.

### Using while loops.

We'll use a while loop to save each of the documents that are left open after we run the Crop and Straighten Photos command. A while loop basically iterates through something while a condition still exists. For example, if I go to McDonald's and order some fries, I will continue to eat a french fry while I still have french fries left on my plate. In Photoshop, we'll save and close open documents until they are all closed. (4_closeAllWithoutSavingCommented.jsx)



```
// While the number of documents is greater than 1
while (app.documents.length >=1){
    // Close the current document without saving
    app.activeDocument.close(SaveOptions.DONOTSAVECHANGES);
}
```

Open some documents Photoshop making sure you don't have any documents open that need saving, and run the script. The script will close each document until there are no documents open. Let's modify this script. We'll delete the lines for alerting the name of the document, since that's just there to demonstrate how the script works. Save the file as closeAllOpenDocumentsWithoutSaving.jsx If you're like me, you'll have dozens documents open sometimes and just want to close all of the files without being asked whether you want to save them or not. Now you have a neat little script to clean up your work area when you are ready to move onto the next project.

### The ScriptingListener plug-in.

If we look in the Object Model Viewer, you'll discover that there aren't methods for every command in Photoshop. You'll also notice that the commands in the COMMANDS panel in Configurator don't cover every command you can perform in Photoshop, just menu commands. Don't distress. That's where the ScriptingListener plug-in comes in handy.

### Installing the ScriptingListener plug-in:

1) Quit Photoshop
2) Locate the ScriptingListener plug-in inside Photoshop's application folder: Adobe Photoshop CS4>Scripting Guide>Utilities>ScriptingListener
3) Copy the ScriptingListener plug-in into Photoshop's Plug-Ins folder: Adobe Photoshop CS4>Plug-Ins
2) Launch Photoshop

### Using the ScriptingListener plug-in:

As you work in Photoshop, the ScriptingListener plug-in records JavaScript for any operation which is Actionable to a log file named `ScriptingListenerJS.log`, which by default is saved to the desktop.

To determine if an operation is Actionable, double-click the ScriptingListenerJS.log file to launch the Console application. Keep the Console window view of the ScriptingListenerJS.log visible as you are working. As you do operations in Photoshop, you will see the log update if the operations you perform are Actionable. Painting, for example, is not an Actionable operation. You can click 'Clear" at any point to clear the entries in the log.

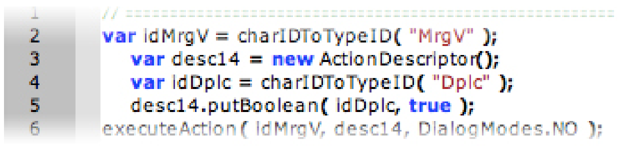
As you will notice, the JavaScript recorded by the ScriptingListener plug-in isn't always easily read or clearly labeled.

Try to get in the habit of recording one step at a time and commenting each operation so you know what it does. I generally record the operation a few times with different settings so I can see where the settings change in the code.

You can also turn the JavaScript that the ScriptingListener plug-in generates into your own functions. You can even pass in your own arguments.

### Stamp Visible and Stamp Visible to Top

The first command we'll capture is the Stamp Visible command which is only available through a little known keyboard shortcut: cmd/ctrl + opt/alt + shift + E. Stamp Visible creates a flattened layer of the current appearance of the document. Open a multi-layer document like Fish.psd from the Samples folder in the Photoshop application folder and cmd/ctrl + opt/alt + shift + E.



```
// =======================================================
var idMrgV = charIDToTypeID( "MrgV" );
    var desc14 = new ActionDescriptor();
    var idDplc = charIDToTypeID( "Dplc" );
    desc14.putBoolean( idDplc, true );
executeAction( idMrgV, desc14, DialogModes.NO );
```

Copy the code from the ScriptingListenerJS.log to the ESTK and run it to test it.

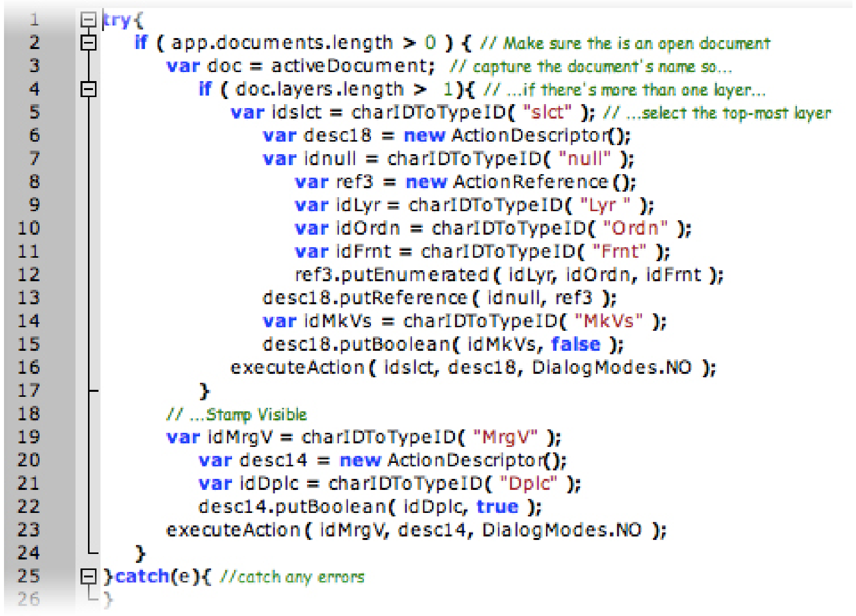
Let's capture some code for Select Back Layer (opt/alt + , ) and Select Front Layer (opt/alt + . ) both of which are insanely useful for scripting and also only available through keyboard shortcuts.

### Using Conditional scripts and Try/Catch to improve the user experience.

Conditionals are a very powerful feature of scripting. You can do an operation if certain conditions are met. A common conditional is to run different actions if the document is wider than it is tall, or if the document is taller than it is wide, using an if/else statement. Other common conditionals check to make sure the document is in the right color mode or bit depth, or the current layer is a text or raster layer.

For our script, we'll use two conditionals. We want to make sure there is an open document open before we run our script to avoid error messages if someone clicks our button to Merge Visible and there is no document open. We also want to make sure the newly merged layer is at the top of our layer stack so we'll use the code for Select Front Layer. We'll want to make sure our document has more than one layer before trying to change the layer selection.

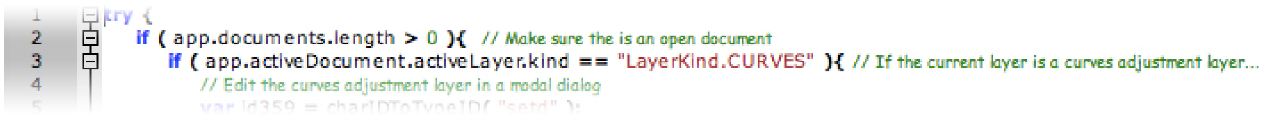
Lastly, we'll wrap everything up in a Try/Catch to handle any errors we may not have thought of. (6_stampVisibleCommented.jsx)


```
try{
    if ( app.documents.length > 0 ) { // Make sure the is an open document
        var doc = activeDocument;  // capture the document's name so...
            if ( doc.layers.length >  1){ // ...if there's more than one layer...
                var idslct = charIDToTypeID( "slct" ); // ...select the top-most layer
                    var desc18 = new ActionDescriptor();
                    var idnull = charIDToTypeID( "null" );
                        var ref3 = new ActionReference();
                        var idLyr = charIDToTypeID( "Lyr " );
                        var idOrdn = charIDToTypeID( "Ordn" );
                        var idFrnt = charIDToTypeID( "Frnt" );
                        ref3.putEnumerated( idLyr, idOrdn, idFrnt );
                    desc18.putReference( idnull, ref3 );
                    var idMkVs = charIDToTypeID( "MkVs" );
                    desc18.putBoolean( idMkVs, false );
                executeAction( idslct, desc18, DialogModes.NO );
            }
        // ...Stamp Visible
        var idMrgV = charIDToTypeID( "MrgV" );
            var desc14 = new ActionDescriptor();
            var idDplc = charIDToTypeID( "Dplc" );
            desc14.putBoolean( idDplc, true );
        executeAction( idMrgV, desc14, DialogModes.NO );
    }
}catch(e){ //catch any errors
}
```

## Using scripting listener to unlock deprecated functionality.

There are some commands that were in previous versions of Photoshop that are no longer exposed in the UI of Photoshop CS4. One example is creating and editing curves adjustment layers in a dialog from CS3. Another example is Merge Linked Layers from Photoshop CS. You can use the ScriptingListener plug-in those versions of the app to record code that you can play in Photoshop CS4. (7_createCurvesDialogCommented.jsx) Note: this is sort of a hack and we can't guarantee this will work in future version of Photoshop.

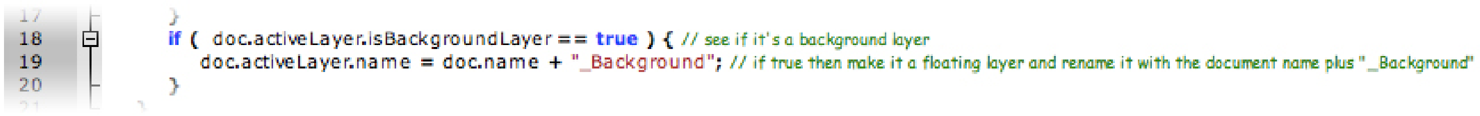
We can also use a conditional to determine if a layer is a Curves Adjustment layer when trying to edit the curve with a script ( 8_editCurvesDialogCommented.jsx)


```
try {
    if ( app.documents.length > 0 ){  // Make sure the is an open document
        if ( app.activeDocument.activeLayer.kind == "LayerKind.CURVES" ){ // If the current layer is a curves adjustment layer...
            // Edit the curves adjustment layer in a modal dialog
            var id359 = charIDToTypeID( "setd" );
```

## Promoting a background layer to a floating layer.

We can use the code we captured for Select Back Layer (opt/alt + , ) to promote background layers in documents. After we select the bottom-most layer, we can check to see if the layer is a background layer using the `isBackgroundLayer` property.

We can also re-name the newly promoted background layer to be the name of the document `doc.name` plus the string `"_Background"` ( 9_promoteBackgroundLayerCommented.jsx)


```
        }
        if (  doc.activeLayer.isBackgroundLayer == true ) { // see if it's a background layer
            doc.activeLayer.name = doc.name + "_Background"; // if true then make it a floating layer and rename it with the document name plus "_Background"
        }
    }
```

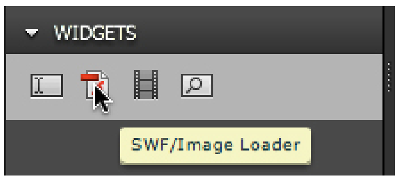
## Understanding the Configurator export process.

When you start to add things like images swf widgets to your panels it becomes important to understand Configurator's Export Panel process. When you create a panel you want to share with other users you should create an empty panel in Configurator and save it before adding any content to it. Configurator project is a .gpc file.

Then create the graphics and swf files you want incorporate into your panel. TIP: Make sure the graphics and swf files are located in the file system right next to the .gpc file from Configurator. This helps maintain relative paths to your assets as you add them to your panel in Configurator. When you do File>Export Panel from Configurator, Configurator will move any referenced assets (images or swfs) to the Plug-ins/Panels folder during the export process, placing everything in a folder called 'content' along with a copy of the .gpc file and a supporting .jsx file, and .res folder. Configurator also creates a .MXI file next to the panel 'content' folder. We'll talk about the .MXI file later on.

## Branding a panel with an image: PanosFX.com Panel.

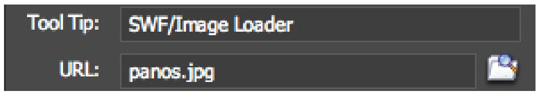
To add an image to a panel, drag a SWF/Image Loader object from the WIDGETS panel in Configurator to your panel.



With the SWF/Image Loader object selected on the panel canvas, click the folder icon after the URL field.

Browse to the image you want to use and click [Select].

If you placed your image next to the .gpc in the file system correctly, you will have a nice relative path with just the name of the file in the URL field:

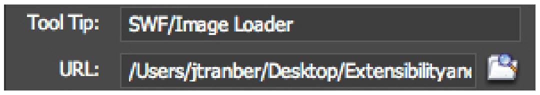


Tool Tip: SWF/Image Loader

URL: panos.jpg

CORRECT!

If you have anything else beside just the name of the image, you have a absolute path that won't work correctly on another users machine:



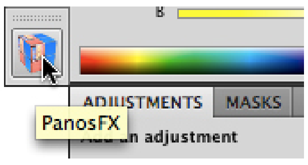
Tool Tip: SWF/Image Loader

URL: /Users/jtranber/Desktop/Extensibilityan

WRONG!

You either forgot to save the panel before adding the image or you didn't place the image next to the saved panel .gpc in the file system.

Export your panel and make sure that the image loads correctly.

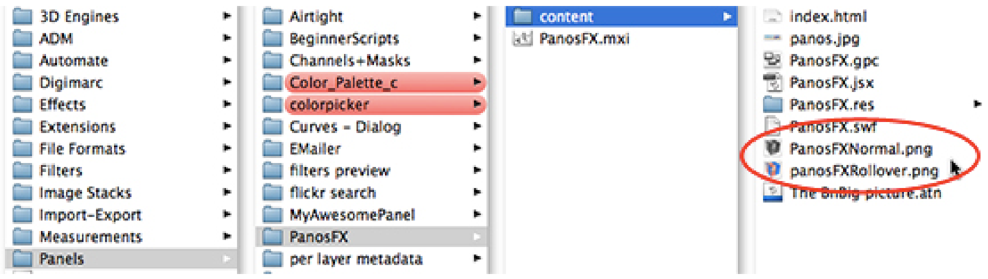


## Adding collapsed panel icons.

You can add icons for your panels that will show up in Photoshop if the panel is collapsed.

Create two 48px by 48px icons in Photoshop, one color and one which has been unsaturated.

Save and name the unsaturated version the same as your panel plus the word normal ("PanosFXNormal.png") and name the color version "PanosFXRollover.png". Place the two .png files in the 'content' folder of your exported panel inside the Plug-ins/Panels folder inside the Photoshop application folder

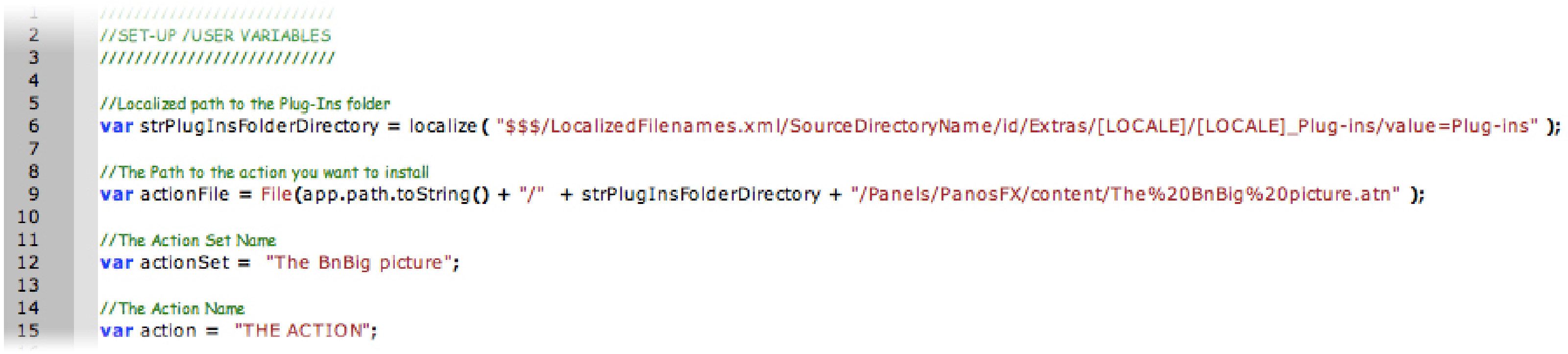


TIP: Configurator won't export the icon files on export so they need to be manually placed in the panel 'content' folder.

## Installing and running an action.

You may want to play an action from a panel. If the panel is for only you to use, you can use the Run Photoshop Action widget to reference an action that is already installed in Photoshop on your computer. If you want to share an action that's played from a panel, it gets a little more complicated.

In Photoshop, select "Save Action..." from the Actions panel menu and save the .atn file to the 'content' folder of your exported panel. TIP: Configurator won't export the action files on export so they need to be manually placed in the panel 'content' folder. By placing items in the 'content' folder we know the items will be installed with the panel when we create an installer using Extension Manager CS4. Extension Manager will be discuss more later on. Next, we need to load and play the action from using a script. (getActionSetsCommented.jsx)



```
///////////////////////////
//SET-UP /USER VARIABLES
///////////////////////////

//Localized path to the Plug-Ins folder
var strPlugInsFolderDirectory = localize( "$$$/LocalizedFilenames.xml/SourceDirectoryName/id/Extras/[LOCALE]/[LOCALE]_Plug-ins/value=Plug-ins" );

//The Path to the action you want to install
var actionFile = File(app.path.toString() + "/"  + strPlugInsFolderDirectory + "/Panels/PanosFX/content/The%20BnBig%20picture.atn" );

//The Action Set Name
var actionSet =  "The BnBig picture";

//The Action Name
var action =  "THE ACTION";
```

Don't be intimidated by the FUNCTIONS section of this script, you only need to worry about the user variables for `actionFile`, `ActionSet` and `action`.

## Add a button that launches a URL

In order to create a button that opens a URL in a web browser we need to reference and execute an external HTML file. When the HTML files is opened it loads the URL http://www.panosfx.com/ (index.html)

```
<head>
    <title>PanosFX.com</title>
    <meta HTTP-EQUIV=REFRESH CONTENT="0; URL=http://www.panosfx.com/">
</head>

<body>
    <p><a href="http://www.panosfx.com/">PanosFX</a></p>
</body>
```

Save the index.html file in the 'content' folder of your exported panel. TIP: Configurator won't export the html file on export so it needs to be manually placed in the panel 'content' folder.

Next is the `file().execute();` command. (openURLCommented.jsx)

```
//Localized path to the Plug-Ins folder
var strPlugInsFolderDirectory = localize( "$$$/LocalizedFilenames.xml/SourceDirectoryName/id/Extras/[LOCALE]/[LOCALE]_Plug-ins/value=Plug-ins" );

//Localized path to the HTML file
var htmlFileName = app.path.toString() + "/" + strPlugInsFolderDirectory + "/Panels/PanosFX/content/index.html";

// Open the HTML file that redirects to the URL
File(htmlFileName).execute();
```

## Create an installer using the MXI file and Extension Manager CS4

When you are finished creating and testing your panel you are ready to create an installer. Configurator automatically creates a .MXI file next to the panel 'content' folder when you export a panel.

content

PanosFX.mxi

To create an installer, double-click the .MXI file. This will launch Extension Manager CS4 which will ask you where to save the installer. TIP: If an older version of Extension Manager launches, delete the old version or open Extension Manager CS4 and manually open the .MXI file. Save the installer to your desktop or another desired location. Extension Manager creates an MXP file. The .MXP file contains everything from inside the 'content' folder plus instructions where to install the panel files.

PanosFX.mxp

To install the panel, double-click the .MXP file. Extension Manager will launch and ask you to confirm if you want to install the panel. TIP: If Extension Manager says you don't have permission to install the files on Windows OS, quit Extension Manager CS4 and relaunch using "Run as Administrator."

## Referencing external scripts.

In order to reference external scripts that live under the File>Scripts menu, you will need to record ScriptingListener code for the scripts you want to reference.
(runAirtightScripts.jsx) Line 4 is simply the name of the script you want to launch.

```
1  var idAdobeScriptAutomationScripts = stringIDToTypeID( "AdobeScriptAutomation Scripts" );
2     var desc2 = new ActionDescriptor();
3     var idjsNm = charIDToTypeID( "jsNm" );
4     desc2.putString( idjsNm, "SimpleViewer" );
5     var idjsMs = charIDToTypeID( "jsMs" );
6     desc2.putString( idjsMs, "undefined" );
7  executeAction( idAdobeScriptAutomationScripts, desc2, DialogModes.NO );
```

## Modifying the .MXI file to install multiple files.

If we want to install scripts that are referenced from our panel, we need to modify the .MXI file. Place the scripts you want to install in a folder next to the .MXI file and the panel 'content' folder. Open the .MXI file in a text editor. In the section called `<files>` we name the `source`, the folder of scripts next to our MXI file, and we name the `destination`, the folder we want to install the contents of the folder into. (Airtight.mxi)

```
    <files>
        <file source="airtight_interactive" destination="$Scripts/" />

        <file source="content" destination="$panels/Airtight/" />
    </files>
```

"$Scripts" is a folder attribute which will work on any language to correctly point to the Presets/Scripts folder inside the Photoshop application folder. For more Photoshop location attributes, see page 14 of the (mxi_file_format.pdf).

## For updated information.

Visit http://www.tranberry.com/panels/