JavaScript / Flash Communication

NAB 16 April 2007

Phillip Kerman (phillipkerman.com LLC)

Avenues of communication

You can think of the Flash/JavaScript as a two-way street: Flash can talk to JavaScript and JavaScript can talk to Flash. If you want an event from the html (say, when the user clicks a traditional hyperlink) to cause your Flash movie to change you'll be going from JavaScript to Flash. If, on the other hand, you want an event in the Flash movie to cause JavaScript to perform some task you're going from Flash to JavaScript. Consider this origin and destination relationship even when one side (JavaScript of Flash) is requesting information back from the other side. That is, you'll always begin in either JavaScript or Flash and trigger the other.

Flash to JavaScript

Flash can trigger both native JavaScript functions (such as `alert()` or `window.open()`) as well as custom functions that you write in the HTML. This means Flash can directly affect the container (that is the browser). You may want to let users click a button in your Flash application that pops open a new browser window at a fixed sized. Or you might want a link to change in the HTML portion of a page when a Flash video has completed.

There are a few ways to send messages from Flash to JavaScript:

1—use the `javascript:` prefix when performing a hyperlink (Flash's `getURL()`)

- 👍 quick and easy
- 👍 works in old versions of the Flash Player
- 👎 gets messy when passing arguments
- 👎 not as elegant as ExternalInterface (below)

2—use Flash's `fscommand`

- 👍 works in *much* old versions of the Flash Player
- 👎 must wrap all JavaScript in DoFSCommand function
- 👎 browsers handle DoFSCommand differently
- 👎 not as elegant as ExternalInterface (below)

3—use ExternalInterface

- 👍 easy and powerful
- 👍 supports multiple data types (the options only pass string data)
- 👍 returns data from JavaScript synchronously
- 👎 requires Flash Player 8 or later

1 Using `javascript:` (Flash to JS)

If you're delivering to Flash Player 8 or later, use ExternalInteface as it's easier and more powerful. If you're targeting Flash Player 7 or lower, this approach is fin. The actual code is easy on the surface. Just use `getURL()` and, in place of the URL, place your entire JavaScript call followed by "javascript:". For example, here we have Flash calling the native JavaScript code that closes the current window (that is, `window.close()`):

```
getURL ( "javascript:window.close()" );
```

You can also trigger custom JavaScript that you have in the HTML. For example, if (in the script tag of your HTML) you have:

```
function myHomemadeThing(){
      //do something here
}
```

Then, from inside Flash you can trigger that function using:

```
getURL ( "javascript:myHomemadeThing();" );
```

This approach gets a tad ugly when you want to pass arguments, for example:

```
getURL ( "javascript:alert('hi mom');" );
```

Notice the nested argument (‘hi mom') had to use single quote marks. It gets worse when you have multiple parameters or those parameters are dynamic. In this next example, I want to call:

```
window.open('other.html', 'winName', 'width=600, height=450')
```

..but I want the filename (`other.html`) as well as the values for width and height to be dynamic based on variables inside my Flash movie. The following code dynamically generates a string then triggers it:

```
var widthVar = 600;
var heightVar = 450;
var filename = "other.html";
var myString = "window.open('" +
   filename +
   "', 'winName', 'width="+
   widthVar + ", height=" +
   heightVar + "')";
//test it:
//trace (myString);
getURL( "javascript:" + myString );
```

Notes:
In this particular case (calling window.open()) it would make more sense to write your own function inside the JavaScript to actually open the window, and simply have Flash trigger the homemade function (as shown in the download files for this session).

Also, there is no reason to attempt this in AS3 because AS3 requires Flash Player 9 and therefore you can use ExternalInterface—which is an easier and more powerful alternative.

2 Using `fscommand` (Flash to JS)

Flash's `fscommand` is the original way to trigger functions in the "container" hosting your .swf. The obvious container is a browser, but it also includes Macromedia Director, shells created in Visual Basic or C++, or any program that can host an ActiveX control. If you're targeting Flash Player 8 or higher, use ExternalInterface. If you're targeting Flash Player 7 or lower, the "javascript:" approach is easier. Why would you even bother with fscommand then? Primarily if you're working in a so-called legacy application or working inside another framework (such as an instant messaging client) that displays Flash in an ActiveX control. Because this presentation is on Flash/JavaScript communication, I think it's safe to say just skip this option. (Here's an overview regardless.)

The code inside Flash *is* a tad easier than the "javascript:" approach. Just use `fscommand()` and pass two paramters (though think of these as a function name followed by any parameters crammed into a single string):

```
fscommand ( "myFunction", "param1,param2,param3" );
```

In ActionScript 3, `fscommand` packaged in `flash.system`, so the above call would look like this:

```
import flash.system.fscommand;
fscommand ( "myFunction", "param1,param2,param3" );
```

In this case, we'll just split apart the three parameters in that parameter upon receipt in JavaScript. As you'll see in the corresponding JavaScript, we won't really be triggering a function called "myFunction" but rather we're always triggering the "DoFSCommand" function—and the first parameter received there is the string "myFunction" (the second parameter received is `"param1,param2,param3"`). Here's *some* of the JavaScript to catch the fscommand call above:

```
function myFlashObject_DoFSCommand ( command, args ){
      switch ( command ){
            case "myFunction":
                  //could see all args as sent
                  //alert ( args );
                  var argsAsArray = args.split(",");
                  for (var i=0; i<argsAsArray.length; i++){
                        alert ( argsAsArray[i] );
                  }
                  break;
            case "otherFunction":
                  break

      }
}
```

Note that the function name follows the convention `myFlashObject` plus `_DoFSCommand`. The `_DoFSCommand` is fixed, and `myFlashObject` part must match the `name` value (in the embed tag for the .swf) and the `id` value (in the object tag). See "Identifying your Flash Object" below because there are a variety of ways to set the value for your .swf's `name` and `id`.

Note too that *all* `fscommand()` calls go through the `myFlashObject_DoFSCommand()` function. I'm using a `switch` statement to sort out the first parameter (`command`) and the `split()` method to turn my sole parameter (`args`) into an array full of strings. Finally, note that even if you only want to trigger native JavaScript functions, if you use `fscommand()` you need to channel everything through the `myFlashObject_DoFSCommand()` function. Naturally, inside this function you can call JavaScript like how I'm calling `alert()` above.

If this wasn't fun enough, we're not out of the woods yet! (I'm just the messenger here.) You need to ensure it works in Internet Explorer too. A .swf running in Internet Explorer it won't trigger `myFlashObject_DoFSCommand`! It will try to trigger a VBScript subroutine (think function) named `myFlashObject_FSCommand` (notice it's missing "Do"). You need to redirect the VBScript function to your "standard" JavaScript function (the one *with* the "Do"). Because of the way VBScript works, you fashion a string and then use `document.write()` to inject it into your HTML. Like this:

```
temp  = '<SCRIPT LANGUAGE="VBScript"\> \n'
temp += 'Sub myObjID_FSCommand(ByVal cmd, ByVal arg) \n'
temp += 'call myObjID_DoFSCommand(cmd, arg) \n'
temp += 'end sub \n'
temp += '</SCRIPT\> \n'
document.write(temp);
```

I've got complete and lucent examples in the downloads for this presentation. Just remember the other ways to have Flash trigger JavaScript are easier.

3 Using `ExternalInterface` (Flash to JS)

Compared to the other masochistic ways to get Flash to trigger JavaScript, ExternalInterface will seem like a dream. What's doubly cool is that ExternalInterface provides an equally eloquent solution when you want to let JavaScript trigger ActionScript—as we'll do later. The three main advantages of ExternalInterface (when going from Flash to JavaScript) are: it's easy, you can get data back from JavaScript synchronously, and you're not limited to sending strings only (you can send numbers and arrays too). Alas, it does require users have Flash Player 8 or above. (Interestingly, when going from Flash to JavaScript the ActionScript 3 code is identical to the ActionScript 2 code.)

Let's start with the code. From your Flash movie, you just trigger the static `call()` method in the class `flash.external.ExternalInterface`. The first parameter is the JavaScript function name, the second and subsequent parameters are the arguments you want to send to the JavaScript function. Sa you want to trigger the native JavaScript function `alert()` as in `alert("hi mom")`, just use this code in Flash:

```
flash.external.ExternalInterface.call("alert", "hi mom");
```

You can easily pass even more parameters, as in the following code to effectively call this JavaScript:

```
window.open('other.html', 'winName', 'width=600, height=450')
```

In Flash you just do:

```
import flash.external.ExternalInterface;
ExternalInterface.call("window.open", 'other.html', 'winName', 'width=600, height=450');
```

Don't let that `import` statement throw you—it simply lets me say `ExternalInterface` instead of `flash.external.ExternalInterface` in every case.

Naturally, you can also call your own custom JavaScript functions. For example, assuming you have a `multipleAlerts()` function in your JavaScript like this:

```
function multipleAlerts ( howManyTimes ){
      for ( var i = 0; i < howManyTimes; i++){
            alert ("alert count " + i );
      }
}
```

…then you can trigger it from Flash like this:

```
flash.external.ExternalInterface.call("multipleAlerts", 3);
```

Notice that I'm also demonstrating that you can pass a number (`3`) as a parameter and it is received as a number.

More about using `ExternalInterface` (namely, getting return values)

While we haven't quite reached the "JavaScript triggering Flash" section yet, it's just so sweet that `ExternalInterface.call()` can get data back from your JavaScript function synchronously (that is, you'll get a result immediately). Naturally, your JavaScript function needs to include a `return` statement. Take this simple JavaScript function:

```
function getDouble ( value ) {
      return 2 * value;
}
```

You can trigger it from Flash using:

```
var result:Number = flash.external.ExternalInterface.call("getDouble", 3);
```

This example is only impressive if you knew the history of how such a maneuver was necessary in the past. As you'll see next, there are a multitude of ways for JavaScript to trigger Flash code. What you'd have to do in the past was trigger the JavaScript from Flash then wait around in Flash for JavaScript to, in turn, trigger some code in the .swf. That is, it was a two step process: Flash says "hey JavaScript send me some info" and then JavaScript would say "hey Flash, here's that info you wanted". The above example is synchronous as the result is returned immediately and it's just way easier than setting up both lines of communication (Flash to JS and JS to Flash).

Perhaps the most common way I've used this feature is to give my .swf access to the query string that arrives in the browser. For example, users could arrive at any one of the following URLs:

```
www.phillipkerman.com/index.html
www.phillipkerman.com/index.html?specialfeature=1
www.phillipkerman.com/index.html?specialfeature=2
```

…but the .swf hosted in index.html wouldn't know the difference. To give my .swf access to that value of the specialfeature variable in the query string I needed to parts: a call from Flash to JavaScript and a JavaScript function that would parse that value and return it. Here's the Flash side:

```
import flash.external.ExternalInterface;
var result:String = ExternalInterface.call("getQueryValue", "specialfeature");
```

Then the JavaScript is a bit more involved:

```
function getQueryValue(match){
      var allPairs = document.location.search.substr(1).split("&");
      for (var i = 0; i<allPairs.length; i++) {
            var index = allPairs[i].indexOf("=");
            var firstPart = allPairs[i].substr(0, index);
            var secondPart = allPairs[i].substr(index+1);
            if (firstPart == match) {
                  return secondPart;
            }
      }
      return undefined;
}
```

While this example lets you search for a specific known value in the query string (`specialfeature`) you could definitely update this code to return an array of values, say, if you didn't know what parameters to expect.

JavaScript to Flash

JavaScript can send messages to Flash and, provided you have some code on the Flash side that's listening, your Flash app can react how ever you want—say, by displaying graphics or playing sound. Perhaps you have a Flash video player that loads different videos based on the traditional html hyperlink a user clicks. Like with the "Flash to JavaScript" options (above), there are a variety of ways to send messages from JavaScript to Flash.

Here's an overview of the ways to send messages from JavaScript to Flash:

1—use methods of the Flash Object

- 👍 works in *much* older versions of the Flash Player
- 👎 limited set of methods
- 👎 limits can mean you need to restructure your Flash application

2—use ExternalInterface

- 👍 easy and powerful
- 👍 returns data from Flash synchronously
- 👍 lets you trigger any method you choose to expose
- 👎 requires that you expose methods explicitly
- 👎 requires Flash Player 8 or later
- 👎 subtle code differences when using ActionScript 3 vs. ActionScript 2

3—use FlashVars

- 👍 easy
- 👍 works in older versions of the Flash Player
- 👍 easily to set initial values
- 👎 can *only* set initial values (nothing at runtime; only when the .swf loads)

The FlashVars option is really out of place here. As you'll see below, it's not really JavaScript. The reason it's out of place is that it's not comparable to the other two options. The FlashVars option is only good for one thing: setting the initial values of variables in your Flash application. It's useful if you want to set the starting value for a variable from inside the HTML (either as a convenient way to set values by hand, or automatically in an .html document that's generated dynamically). I included it here because you are sending values from HTML to Flash. Finally, when appropriate, this is the best option. Just don't think of this as an obsolete feature.

Identifying your Flash Object

Before I detail the ways to send messages from JavaScript to Flash I want to discuss how your HTML and JavaScript will identify your Flash object (that is, the Flash element in the web page). Basically, you just need to give each .swf in your web page an identifier. The need to identify the Flash object comes up in several places: when using `fscommand()` to trigger JavaScript from Flash (above) the actual function name in the JavaScript must begin with your Flash object's name; when triggering methods of the Flash Object class you must refer to the specific Flash object instance name (as your web page could have more than one .swf present); and, when using ExternalInterface to trigger methods in your Flash application, you must refer to the specific Flash object instance name. In summary, when you're in JavaScript and want to talk to Flash, you have to address the specific .swf and you do that by using its object name.

I keep saying "the Flash object's name" but realize it's called "name" in the embed tag and called the "id" in the object tag—you have to set both. In the following html generated from Flash 8's publish command you can see the "myObjectID" identifiers (name/id) appearing in two places shown in bold below:

```
<object classid="…" codebase="…" width="550" height="400" align="middle"

      id="myObjectID" >

      <param name="allowScriptAccess" value="sameDomain" />
      <param name="allowFullScreen" value="false" />
      <param name="movie" value="filename.swf" />
      <param name="quality" value="high" />
      <param name="bgcolor" value="#ffffff" />

      <embed src="filename.swf" quality="high" bgcolor="#ffffff" width="550" height="400"

      name="myObjectID"

      align="middle" allowScriptAccess="sameDomain" allowFullScreen="false"
      type="application/x-shockwave-flash" pluginspage="…" />
</object>
```

Actually, when you use Flash 8 to publish, it will use "movie" for your name/id.

If you're using Dreamweaver 8 to create the HTML, you can simply use the Properties panel to set the name/id as shown here:

The only problem with the code generated from Flash 8's publish command or Dreamweaver (8.0.1 or earlier) is that when visitors arrive at your page using a newer version of Internet Explorer, they'll have to click once before interacting with your application—they'll just see the tool tip "Click to activate and use this control". This is the so-called "Active Content" issue (for more, see `www.adobe.com/devnet/activecontent`). The good news is that Macromedia/Adobe has addressed this issue with a script file "AC_RunActiveContent.js". When using Dreamweaver 8.02 or later, and provided your preferences for the category "Code Rewriting" is set to "Insert using broswer-safe scripts" you don't have to do anything special—whatever you set using the Properties panel will get sent to the AC_RunActiveContent.js script and everything should work.

If you're publishing with Flash CS3 you'll need to set the name/id values in a different place, shown here:

```
if (AC_FL_RunContent == 0) {
      alert("This page requires AC_RunActiveContent.js.");
} else {
      AC_FL_RunContent(
            'codebase', '…',
            'width', '550',
            'height', '400',
            'src', 'filename',
            'quality', 'high',
            'pluginspage', 'http://www.macromedia.com/go/getflashplayer',
            'align', 'middle',
            'play', 'true',
            'loop', 'true',
            'scale', 'showall',
            'wmode', 'window',
            'devicefont', 'false',
            'id', 'myObjectID',
            'bgcolor', '#ffffff',
            'name', 'myObjectID',
            'menu', 'true',
            'allowFullScreen', 'false',
            'allowScriptAccess','sameDomain',
            'movie', 'filename',
            'salign', ''
            );
}
```

If you then use Dreamweaver (8.0.2 or later) to modify the HTML, you can still use Dreamweaver's Properties panel to change the name/id in one convenient place.

Finally, a highly recommended alternative to using either Dreamweaver or Flash's Publish command is to use SWFObject (`blog.deconcept.com/swfobject`). There are a ton of advantages that I won't discuss here, but when it comes to specifying the name/id, it's simply the second parameter:

```
var so = new SWFObject("filename.swf", "myObjectID", "550", "400", "9", "#FFFFFF");
so.write("flashcontent");
```

The code above appears in a script tag under a div with the id flashcontent (which gets replaced by the .swf). Also, the swfobject.js file needs to be referenced by your html. In Pseudo

Once you've set the name/id you'll be able to refer to the Flash object this name. In order to receive `fscommand()` calls from Flash, you name the function `myObjectID_DoFSCommand()`. When triggering one of the support methods of the Flash object you effectively do `myObjectID.supportedMethodName()`. When using ExternalInterface to trigger functions that you've exposed in your Flash movie, you'll effectively do `myObjectID.myFunction()`. Unfortunately, these last two examples were pseudo code. In Internet Explorer, the JavaScript to reference a Flash object with the name/id `myObjectID` looks like this:

```
window["myObjectID"]
```

In other browsers it looks like this:

```
document["myObjectID"]
```

To make this issue easier, I've written the following function so that, in fact, *whenever* you want to reference `myObjectID` you simply say `thisMovie("myObjectID")`:

```
function thisMovie( movieName ) {
      if (navigator.appName.indexOf("Microsoft") != -1) {
            return window[movieName];
      } else {
            return document[movieName];
      }
}
//so the call looks like: thisMovie("myObjectID").supportedMethodName();
```

In summary, you need to give your .swf an identifier (that is, the name/id). Then, when referring to it, you can use a function such as the `thisMovie()` function shown here. Now, back to the ways to send messages from JavaScript to Flash.

1 Using methods of the Flash Object (JS to Flash)

This approach works in older Flash Player versions, but is limited to just a few methods. Instead of listing the supported methods, it's easiest to simply just use two: `SetVariable()` and `TCallLabel()` in a specific strategy that I'll explain next. That is, instead of using `Play()`directly from JavaScript (one of the support methods), use `SetVariable()` and `TCallLabel()` to tell Flash to tell itself to `play()`. I'm just saying that you can effectively replace all the methods with the strategy I'm about to describe. (For the whole list, check out the table on page 3 of my ancient article at `http://phillipkerman.com/articles/muj_84_javascript.pdf`)

There are two basic approaches I have taken to receive events from JavaScript. The approach that uses SetVariable() involves JavaScript setting a variable in the Flash application. But just having a variable in Flash change isn't enough—you need to be noticed when it changes. So, the JavaScript can follow the SetVariable() call with a call to TCallLabel(). So, what happens is that JavaScript sets a variable and then causes a clip in Flash (or the main timeline) to jump to a certain label. Consider a JavaScript function that lets users choose "red", "green", or "blue". The JavaScript could look like this:

```
function goColor (whatColor){
      //set the var
      thisMovie("myObjectID").SetVariable("currentColor", whatColor);
      //now tell Flash to update:
      thisMovie("myObjectID").TCallLabel("flash_0/", "myUpdateNowFrameLabel");
}
```

In Flash, you'd just have to have a frame with the label "myUpdateNowFrameLabel" and in that code you might do something like this:

```
myGoColorFunction ( currentColor );
```

The point is that Flash knows when the variable has changed because it's forced to call (that is, go to and return from) a particular frame.

2 Using `ExternalInterface` (JS to Flash)

While ExternalInterface really does make things easier than alternatives, using it to let JavaScript talk to Flash is a bit more involved than going the other way (Flash to JavaScript). However, you can use it to do more than just "talk" to Flash—you can actually trigger any function in your Flash app, and alternatively get return values too.

The process involves first exposing specific functions by name inside your Flash application. You use `addCallback()` to both point to a Flash method and give it a name by which JavaScript can access it. Say you have a method inside Flash called `displayStringFromJS()`. That Flash method may look like this:

```
function displayStringFromJS ( whatString:String ){
      myField.text = whatString;
}
```

If you want JavaScript to be able to call this function, you'd add this to your Flash code:

```
import flash.external.ExternalInterface;
ExternalInterface.addCallback ("displayStringFromJS", null, displayStringFromJS);
```

The first parameter is the name by which JavaScript will access this method. (You can change that to whatever you want—just remember when you call it from JavaScript to use that name.) The second parameter (currently `null`) gives you a way to specify the scope in which the Flash method will get called—that is, you can control what `this` will resolve to when the function is called (it's a important features and you can leave it at `null`). Finally, that last parameter is a reference to the method you defined earlier.

In ActionScript 3, you leave out that second parameter:

```
import flash.external.ExternalInterface;
ExternalInterface.addCallback ("displayStringFromJS", displayStringFromJS);
```

The code on the JavaScript side looks like the previous section ("Using methods of the Flash Object"):

```
thisMovie("myObjID").displayStringFromJS("hello from JS");
```

This example doesn't show it, but you can send other data types from JavaScript to Flash such as Array and Number. In addition, you can easily (and synchronously) return data back to JavaScript if your method returns a value. Simply add `return` to your method!

3 Using `FlashVars` (HTML to Flash)

Discussion of FlashVars doesn't really belong in paper on JavaScript/Flash but I think it's worth mentioning as an alternative. It's really easy and useful when appropriate. Basically, you can set the initial value for any variable in your Flash application. It's sort of like how you can add variables to the query string *of the .swf embed*. That is, instead of saying `filename.swf` you say `filename.swf?myvar=initvalue`. When you start the Flash app you can check the value of `myvar` to see it is set to `"initvalue"`. Note that this approach adds the additional variables to the .swf not the html. It's sort of messy adding the variables to the .swf filename however.

The FlashVars approach is similar but easier to read because it's more explicit. In the standard object/embed tag you add the following to elements:

```
//in the object tag:
<param name="FlashVars" value="myvar=my init value">
//in the embed tag:
FlashVars="myvar=my init value"
```

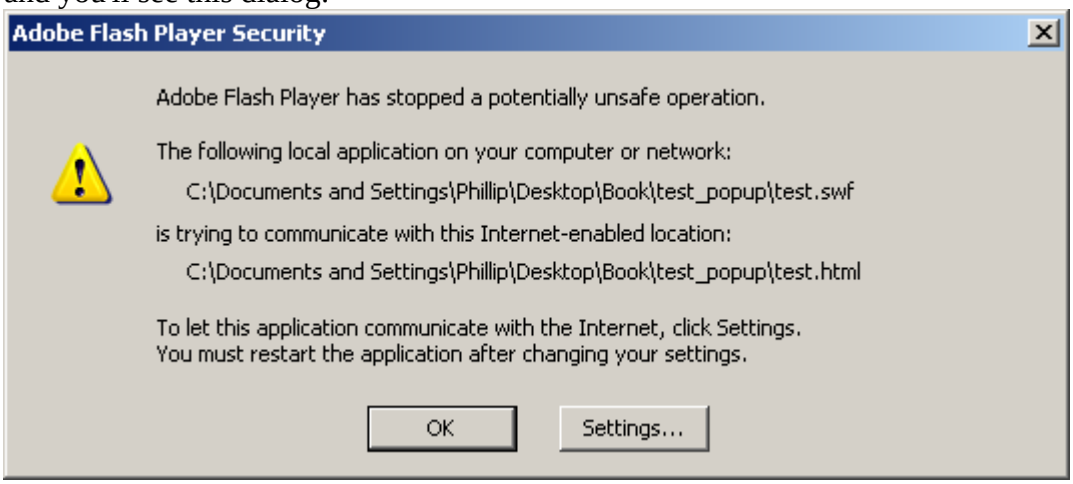
If you're using the Active Content version of publish (see notes earlier), then just add it in one place:

```
AC_FL_RunContent ( 'FlashVars', 'myvar=my init value',   …followed by the rest
```

You can add additional variables using URL encoding—that is, separating additional values with ampersands (&).

SECURITY

When testing pretty much any example from this paper locally, you may run into the Flash player's local security "features". (First, they really are features because you don't want a .swf that you download to have access to your local hard drive and network and then also have the ability to send data out to the internet.) If you post the samples from this paper online, they should all work. However, if you run them on your hard drive many will fail and you'll see this dialog:



If you click "Settings…" you can set permissions for future sessions. The url is:
`www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04a.html`
From there you can explicitly identify specific local files or local directories that you want to grant permissions to read locally.

You can also create a configuration file and list any files or folders to which you want to grant permissions. Just create a text file containing the full path to the folder you want to allow, and place that file in the following folder:
C:\Documents and Settings\Phillip\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust
(replace " Phillip" with your username)
On Mac use:
/Users/Phillip/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust

Related Technologies

While I'd love to go on and on, I won't. However, there are a few other technologies that I'll point to now.

LocalConnection lets you send messages between 2 or more .swfs running on the user's machine. This includes popup windows. So, for example, when the user clicks in a Flash application in one window it could make a change to the content in the other window.

Local SharedObject lets you write data to disk and read it at subsequent sessions. Basically, this is Flash's "cookies". But, unlike cookies you can store other data types besides String. Realize, if the user moves to a different machine you can't access the data you saved on the first machine.

XML is sort of a huge topic and one that I'm presenting on next. See the associated paper I produced for more.

Javascript integration kit was produced by employees at Macromedia to provide some of the same benefits that ExternalInterface offers. However, it was written to work in Flash 6 and later. Check it out at:
`www.macromedia.com/go/flashjavascript/`