AcroTeX.Net

AcroTeX PDF Blog

Scripting Bridge

ActionScript to JavaScript

D. P. Story

The source files for this AcroTeX PDF Blog are attached to this PDF: `AlertConsoleTst.mxml` is the MXML source file for the SWF file of this article, and `AlertConsoleTst.swf` is the SWF file. The Adobe Flex SDK was used to compile `AlertConsoleTst.mxml` to obtain `AlertConsoleTst.swf`.

AcroTeX PDF Blog #3

http://www.acrotex.net
Published: October 9, 2008

Table of Contents

1. Introduction

AcroTEX PDF Blog #3 continues the discussion of the rich media annotation, a new feature of version 9 of the Acrobat application. The *Scripting Bridge* represents the two-way communication between a Flash application and Acrobat. The primary focus of this article is on the communication link from the Flash application (SWF file) to Acrobat (PDF file).

The scripting language of Adobe Flash is ActionScript, and for Acrobat it is JavaScript; consequently, this article demonstrates how ActionScript can be used to communicate with, pass data to, and execute functions within Acrobat.[1]

2. ActionScript to JavaScript

An Adobe Flash application can call two types of JavaScript:

- **Top-level JavaScript (global) functions.** Of these, the most useful is the `eval()` function.[2]
- **Document JavaScript functions.** Functions defined at the document-level can be called from within a Flash application. The document JavaScript can be viewed/edited through the menu `Advance > Document Processing > Document JavaScripts`.

As mentioned above, the `eval()` function is most useful because it gives us access to all the JavaScript for Acrobat API. For example, executing

```
eval("console.println('Hi World')")
```

from the Flash application writes this classic message the the JavaScript Debugger Console window.

This PDF file contains a document JavaScript that will be called by the Flash widget on the next page. The verbatim listing is

```
function myAcroAlert(obj) {
    var rtn=app.alert(obj);
    return rtn;
}
```

[1] When viewing or executing script, Acrobat and Adobe Reader have the same functionality, unless otherwise noted. Acrobat Pro (Extended) is used to create a rich media annotation through its user interface. Adobe Reader cannot create rich media annotations.

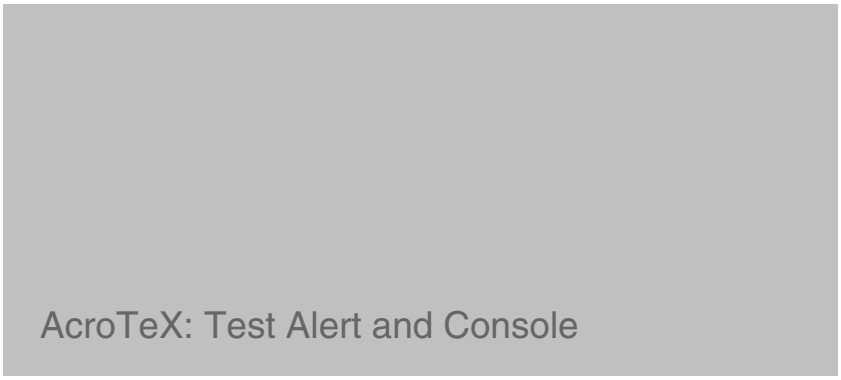
[2] See the *Core JavaScript 1.5 Reference* for information on the standard global objects and functions at the Mozilla Developer Center (http://developer.mozilla.org/en/Core_JavaScript_1.5_Reference).

Note that myAcroAlert takes an object as its argument. This object is passed to the app.alert method. See Acrobat for JavaScript API Reference for more information on the app.alert method. Basically, the object passed is of the form

```
{cMsg: <string>, nIcon: <number>, nType: <number>, cTitle:<string>}
```

The `cMsg` key is used to set the message of the alert, `nIcon` is used to select the icon that is displayed, `nType` is used to set the buttons that appear on the alert dialog box, and `cTitle` is a sting that appears in the title.

The rich media annotation displays the `AlertConsoleTst.swf` Flash application.



When you click the **Console** button (a Flash button, not an Acrobat forms button), the text that appears in the text field is written to the console window of Acrobat. When you click the **Alert** button, the document function `myAcroAlert` is called from the Flash application. The alert message is the text that appears in the text box. Note that the icon and the number of buttons that appear in the alert dialog box change as the alert button is pressed again and again. When one of the buttons is pressed on the alert dialog box, the return value of app.alert is a number that

2.1. Communicating with `ExternalInterface.call`

The Flash application developer can call an JavaScript function from the SWF widget using the `call` method of the `ExternalInterface` class[3]

```
var retn:Object=ExternalInterface.call(
        functionName:String,...arguments);
```

To bring the `ExternalInterface.call` function into your SWF file, you need to execute `import flash.external.*`, or, `import flash.external.ExternalInterface`,

[3] http://livedocs.adobe.com/flex/3/langref/index.html

within the mx:Script tag.

In the next two sections, we describe the use of ExternalInterface.call and how it was use in the AlertConsoleTst.swf file. The reader should consult the source file AlertConsoleTst.mxml for a complete listing.

• The Alert button

When the Alert button is pressed, the following function is executed from within the SWF file. This function calls the document JavaScript function, myAcroAlert, listed earlier in this blog, and also calls the eval function to report the return value to the console.

```
private function myAlert():void {
```

currentIcon and currentType are integer value variables, we perform mod 4 arithmetic to cycle through the values of the nIcon and nType key.

```
    currentIcon = ++currentIcon % 4;
    currentType = ++currentType % 4;
```

We populate the alertMsg object with the argument parameters. The value of cMsg is myText.text, this is the text that appears in the text field between the two buttons.

```
    var alertMsg:Object = {cMsg:myText.text, nIcon: currentIcon,
        nType: currentType, cTitle:"AcroTeX"};
```

Execute ExternalInterface.call with the string "myAcroAlert" as the first argument, with the list of parameters following. Here the only parameter is the object literal alertMsg. The return value is saved in result

```
    var result:Number = ExternalInterface.call("myAcroAlert", alertMsg);
    var respStr:String;
```

Depending on the value of result, we build a string to write back to the Acrobat Debugger Console window.

```
    switch (result) {
        case 1: respStr = "You pressed the \"OK\" button"; break;
        case 2: respStr = "You pressed the \"Cancel\" button"; break;
        case 3: respStr = "You pressed the \"No\" button"; break;
        case 4: respStr = "You pressed the \"Yes\" button"; break;
        default: respStr = "I'm not sure what you did!";
    }
```

Finally, we execute ExternalInterface.call, with "eval" as the first argument. For the second argument, we use "console.println('"+respStr+"')". The eval function in

Acrobat, executes its argument, thus, `console.println` with the `respStr` as its argument is executed and `respStr` is written to the Console window.

```
    var retn1:Object=ExternalInterface.call(
        "eval", "console.println('"+respStr+"')");
}
```

• The Console button

The function `myConsole()` is executed when the **Console** button is pressed is very simple. The verbatim listing follows:

```
private function myConsole():void {
    // show Acrobat JavaScript Console
    var retn1:Object=ExternalInterface.call( "eval", "console.show()" );
    var retn2:Object=ExternalInterface.call( "eval",
        "console.println('"+myText.text+"')");
}
```

The function first opens the console window, this is needed for users of Adobe Reader where there is no user interface to the console window. Next, we execute `eval` again, this time to write the value of text field to the Console.

3. One problem

When I sent this blog to my friend Jürgen, the first thing he tried was to type is name into the text field of the `AlertConsoleTst` widget. He loves his ü (u-umlaut). The **Console** button worked well, it wrote "Jürgen" to the Console window, the result was not a good one with the **Alert** button. Try entering Jürgen name, I used unicode, `J\u00FCrgen` which worked for the **Console** button, for the **Alert** button, the alert message was `J\u00FCrgen`. I need to do more research into this problem, perhaps one of you out there in the PDF/Flash/FLEX community already knows the answer? What say you?

The next AcroTeX PDF Blog will cover JavaScript to ActionScript communication, but for now, I simply must get back to my retirement. ☺