

## Annotations from jkingFormsTut.pdf

### Page 1

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:29:48 AM*

Welcome to this short introduction to the use of JavaScripts in Acrobat Forms. Each slide has an annotation like this one that you can read to get some of what an oral presentation provides.

You might want to use the "Tools->Annotations-> Summarize Annotations" to get all the annotations and then print them out. In that way you can read them and still see the full slide unencumbered.

Thanks for your interest.

--Jim King 2/10/00

### Page 3

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:11:06 AM*

First we set the stage and then show several examples. Examples are the easiest way to show how to use JavaScript in Acrobat Forms.

### Page 4

---

*Annotation 1; Label: jking; Date: 02/10/2000 4:18:09 AM*

Note that "JavaScript" and "Java" are two very distinct languages. JavaScript was invented by people at Netscape and is a scripting or interpreted language in the spirit of PERL or BASIC.

The Acrobat PDF viewers have within them the ability to read and execute JavaScript programs. Special features have been added to this version of JavaScript by Adobe to make it easy to control and modify Acrobat forms.

### Page 5

---

*Annotation 1; Label: jking; Date: 02/10/2000 4:20:58 AM*

You don't really have to be a programmer to be able to benefit from JavaScripts in Acrobat Forms. Some powerful things can be done very simply without getting into sophisticated programming.

By far the easiest way to do JavaScripts in Acrobat forms is by taking an example, like the ones I am providing here, and modifying them for your particular needs. It is easy to make trivial mistakes like leaving the "console." off of "console.println()". Cutting and pasting from working examples avoids that class of mistake.

### Page 6

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:12:31 AM*

JavaScript usage within Acrobat Forms is well documented, at least from a reference manual viewpoint. The two key documents ship with Acrobat and can be found in the Help directory as indicated on this slide.

### Page 7

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:15:41 AM*

This is a simple example of how to present an instruction or warning just BEFORE the person begins to fill out a particular field. We use the "app.alert" function. Read about "App Object Methods" in the AcroJS.pdf document in the Help folder of Acrobat. See page 13 for a description of "alert". This example does not use the optional 2nd and 3rd arguments to the alert method but just supplies a string indicating the message to display.

This JavaScript is entered as an "Action" and as a "Mouse Up" script activated when the mouse button is released selecting this field for entry.

Using the viewer toolbar activate the "Forms Tool" which will expose all the form fields on the page. Double click on the "FieldA" rectangle to open the field editing dialogue. Then select the "Actions" tab and highlight "Mouse Up" and "JavaScript" and then click on "Edit ...". This will expose the JavaScript code to you. This is the window where the JavaScript code is entered and changed.

Notice that it begins with "app.beep(0)" which is documented on page 13 of AcroJS.pdf. This makes an alert sound before the alert is displayed to bring attention to it. The "0" argument indicates which kind of a beep sound to make.

Note: there are 4 forms fields on this page: FieldA, FieldA-JavaScript, FieldA-JavaScript-hide, and FieldA-JavaScript-show. The last three are just for the presentation. I didn't want the JavaScript to show until I was to talk about it. So I made a field "FieldA-JavaScript" that had the text I wanted to show later and I initially set its attributes to not show it on the page. Then I made the invisible field "FieldA-JavaScript-show" which when clicked on will unhide the FieldA-JavaScript field. Finally I added a FieldA-JavaScript-hide so that I could re-use the slides without editing the fields by hand to make the description hidden again.

## Page 8

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:17:50 AM*

Click on Field B and it will ask you if you are happy or not. Depending upon which answer you give you will get a happy face or a sad face.

This example shows a further use of the "app.alert" method described on page 13 of the AcroJS.pdf manual. This time we use all three parameters and also check the return code from the function to find out which answer the user gave (yes or no).

Following the instructions provided on the previous page you can examine the JavaScript associated with the Mouse Up Action on FieldB on this page.

The code "var happy = app.alert("Are you happy?", 3, 2);" first defines a local variable "happy". It then calls the alert function. The "3" in the call indicates that this is a "Status" type call and the "2" indicates that the alert should display yes and no buttons for the response. See page 13 of the AcroJS.pdf document for more options on the alert function. When the function returns it sets "happy" to either 4 or 3 depending upon whether the user clicked on yes or no, respectively.

The second lesson of this example is using JavaScript to access and set attributes on other fields in the document. In this case we have two "button" fields on the page called "HappyFace" and "SadFace" which are in nearly the same spot to the right of FieldB. I made two simple faces using Adobe Illustrator, one happy and one sad, saved them as PDF files and then used them to be the "Icon Only" displays for these two button fields. Initially both buttons are "hidden" and not displayed.

Returning to the JavaScript of FieldB, next two more local variables are defined "h" and "s" which stand for "happy" and "sad". These are then assigned the association with the Fields "HappyFace" and "SadFace" using the "this.getField" function. Thus the variable "h" now refers to the field named "HappyFace" on this page. All of the JavaScript-accessible properties of fields are described starting on page 36 of the AcroJS.pdf file under: Field Properties. In this case we use the field property "hidden" which controls whether or not the field is shown on the page. "h.hidden" refers to the hidden property of "h" which in this case is the "HappyFace" field. We then set it to either "true" or "false" depending upon whether we want it to show or not. We do the opposite settings for the field "SadFace" using the "s" variable.

## Page 9

---

*Annotation 1; Label: jking; Date: 02/14/2000 5:58:38 PM*

To see what this example is about just move your mouse over the FieldC. You should see the colors of the filled area, the outside stroke and the text change.

This example makes use of the "Mouse Enter" and "Mouse Exit" actions associated with "FieldC". To see the JavaScripts open the FieldC definition, click on the "Actions" tab and then highlight "Mouse Enter" and "JavaScript" and click "Edit ...". You can do the same for "Mouse Exit".

In both cases we define a local variable "f" to refer to ourselves, "FieldC", and set the "fillColor", "strokeColor" and "textColor" properties of "FieldC". Again look to page 36 of the AcroJS.pdf manual found in the Acrobat "Help" folder to see about the properties of fields accessible from JavaScript.

Named colors are provided as "color.yellow", "color.blue" etc. as found on page 18 under "Color Properties" in the AcroJS.pdf manual. The "Mouse Exit" JavaScript specifies the colors with "Color Arrays" which allow arbitrary RGB and CMYK values to be specified (each number ranges from 0-1). The description of them is found on page 17 of the AcroJS.pdf. Note that 0.2 for Magenta in a CMYK specification means use 20% of the full ink, whereas 0.2 for Red in an RGB specification means turn on the red gun in the CRT to a 20% level. Thus (0, 0, 0) for RGB means black or no energy in the CRT guns, but (0, 0, 0, 0) for CMYK means no ink or white paper.

## Page 10

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:21:31 AM*

This example shows how to do more complicated calculations using JavaScripts. Enter values in the cost and quantity columns and see that the Total field tracks with the correct sum.

To find the JavaScript open the "Total" field and click on the "Calculate" tab. There you will see that the "Custom Calculation Script" selection has been darkened and you can examine the script by clicking on the "Edit ..." button.

What the script does is compute the sum of products:  $\text{Cost.0} * \text{Quantity.0} + \text{Cost.1} * \text{Quantity.1} + \text{Cost.2} * \text{Quantity.2} + \text{Cost.3} * \text{Quantity.3}$  as the result of the value for the field "Total".

This is shown as a loop through "N" (4) cases one for each line of the file. (For  $i=0$  to  $<N$ .) See page 77 of the JSSpec.pdf file found in the Acrobat "Help" directory for the definition of the "for" Statement. I show this example using a for loop so that you can use it for cases where you might have 16 or 20 lines. Just change "var N=4" to be "var N=16;" or whatever. The script calculates the field names "Cost.0", "Cost.1" etc. by concatenating (using "+") the variable value "L1" which has been set to "Cost" with a "." and with the value of "i" which goes from 0 to 1 to 2 to 3 as the for loop is executed 4 times. You can see this in the statement "var tok1 = L1 + "." + i;". Similarly for "tok2" using "Quantity".

The "this.getField()" function is not used to set a variable but to directly access the "value" of the field whose name is now in the variables "tok1" and "tok2".

The sum is accumulated in the variable "sum" with the "+=" assignment which adds the right side to "sum"; After the loop is finished we assign "sum" to the value of this field "Total".

## Page 11

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:22:54 AM*

There are different levels of scripts depending upon how wide you want their scope to be. Thus the JavaScripts used in "actions", "validate" and "calculate" tabs of fields are just for those fields.

However one can define scripts and functions that have the scope of the whole document (rather than just one field) by entering them in the "Document JavaScripts" repository accessed as shown on the slide. If you go there you will see that we have set up two document JavaScripts called "ChangeAllBackgrounds" and "SetupGlobals" which we will talk about next. You can examine the scripts by highlighting their name and then clicking on "Change". Just click on "Done" when you want to close the overall window. Scripts defined in the "Document JavaScripts" repository are written out with this file, as are the scripts defined within each field. So when the document is sent to someone else those scripts go with it.

It is also possible to define scripts that are available for all documents used on the computer on which the scripts are installed. They are put into a "JavaScript" folder in the Acrobat Folder as indicated on the slide. We won't say any more about them here but you can find out more in the AcroJS.pdf manual on page 56 under "Implementation considerations".

## Page 12

---

*Annotation 1; Label: jking; Date: 02/14/2000 11:25:45 PM*

This example shows a simple "Actions" on the "Mouse Up" of the button "ExecuteDocScript", but it has dramatic consequences. When you click on the first and visible "Push Me!" button the background color of every field in the document is changed to a golden color instead of the transparent color that is the default.

After opening the button, click on the "Actions" tab and highlight "Mouse Up" then highlight "JavaScript" to see the script. Actually there are two demonstration buttons on this page, the first called "ExecuteDocScript" and the second called "ExecuteDocScript-reset". The second one recovers the damage done by the first. The second button is made visible when the first is clicked by another "Show/Hide Field" "Actions" that is displayed after the JavaScript one.

The second button made visible when pushing the first sets the color of all backgrounds back to transparent using a function defined at the document level and found in the "Tools->Forms->Document JavaScripts ..." repository.

But first look at the JavaScript of the Mouse Up of "ExecuteDocScript". It makes use of "this.numFields" a "Document Object Property" the value of which is the total number of fields in this document. See page 22 of the AcroJS.pdf manual. It also makes use of the "this.getNthFieldName(i)" function that returns the name of the i-th field. See page 25 of AcroJS.pdf.

The reset action uses the same method but by defining a change backgrounds function first in the Document JavaScripts and then calling it from the Mouse Up Actions of the ExecuteDocScript-Reset field.

## Page 13

---

*Annotation 1; Label: jking; Date: 02/14/2000 11:36:12 PM*

Similar to categories of JavaScripts there are also categories of variables used in those JavaScripts. Variables local to the JavaScript are used are declared using the "var" declaration. One can define variables that have a scope of the whole document by using the "this" qualifier that usually refers to the Document. So "this.y" is a variable that is common across all JavaScripts in the document. So if "this.y" is set to "3" in one JavaScript it can be next accessed in another JavaScript and it will still have the value "3".

It is also possible to have variables that have the scope across all documents in an Acrobat "session". This is done by using the "global.setPersistent( .. , false)" function. See page 50 of the AcroJS.pdf file under "Global Object". For variables that maintain their value across Acrobat executions you can make them persistent by using "true" instead of "false". In this case they are saved in the "glob.js" file

in the Acrobat Plug-Ins directory.

## Page 14

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:04:37 AM*

Enter a value in the "Set All:" field and see which of the other fields change. In the four fields on the left there are both "validate" and "calculate" JavaScripts defined for each of them. The FieldE\_var field defines its result as a local variable and so setting a variable of the same name in the "FieldE\_setall" field has no effect. The "FieldE-this" field defines its result as a document (this) variable. Setting a "this" variable of the same name in the "FieldE\_setall" defines the same variable and so there is a change.

In a "SetupGlobals" JavaScript defined at the document level in the "Tools->Forms->Document JavaScripts ..." repository several get and set functions are defined. They use the "global.setPersistent()" function described on page 50 of the AcroJS.pdf file.

The "FieldE\_global\_false" field calls the set-functions that have "false" in the "setPersistent" function call. The "FieldE\_global\_true" field calls the set-functions that have "true" in the "setPersistent" function call.

If you really want to test this thoroughly you can extract the extra page at the end (Page 16) and save it as a separate file. Then set a new value into the "Set All:" field and go to the separate file and see which values change. Only the "FieldE\_global\_true" and the "FieldE\_global\_false" fields will track across documents. In order to get the second document to re-evaluate the field entries you need to alter one of the fields.

If you quit Acrobat and restart it you will find that values in only the field "FieldE\_global\_true" will live across the restart.

## Page 15

---

*Annotation 1; Label: jking; Date: 02/15/2000 9:28:49 AM*

Thanks for looking through these examples. There is one extra page after this one to be used in conjunction with the example on the previous page. See the notes on those pages to understand what that extra page is for.

Jim King -- 2/10/00

I hope you have found this all in-form-ative and form-filling.

## Page 16

---

*Annotation 1; Label: jking; Date: 02/15/2000 12:08:19 AM*

This extra page is to help understand the various kinds of variable as shown on Page 14. Use the "Document->Extract Pages ..." tool to make a separate PDF file of this one page. Then set values into the "Set All:" field on Page 14 to see which fields in this one page file change. Then do the same thing but quitting the viewer in between the set and examination. You will see that there is a way to save a value across Acrobat sessions.