



AcroTeX.Net

AcroTeX PDF Blog

Processing Acrobat Forms using JavaScript

External Processing of a Field

Part 8: Processing Collections of Fields

D. P. Story

The source files for this AcroTeX PDF Blog are attached to this PDF:

- [blog19_scripts.zip.txt](#) is a zip file that contains the server-side scripts `apb_tabsave_fdf.asp` and `apb_tabsave_html.asp` used for the survey in this article.

Both scripts use vbscript as the scripting language, and require a windows server, e.g., Microsoft IIS. The ASP file `apb_tabsave_fdf.asp` requires the FDF Toolkit.

Save the file `blog19_scripts.zip.txt` as `blog19_scripts.zip`, then unzip.

Table of Contents

11 Processing a Collection of Fields	3
11.1 Creating a Survey Form	4
11.2 Check Required Fields	5
11.3 Submit a Single Stream	8
11.4 Submit as HTML	10

11. Processing a Collection of Fields

In blogs #12–#18, we covered the techniques of how to extract information from a form field and how to change the appearance of a form field programmatically using JavaScript. The script, usually activated by a button, acted on its various brother and sister fields. I refer to this as *external processing*, the form field itself is not getting, setting, and interpreting itself, these tasks are done by another field, one external to itself. We meticulously surveyed all fields—text fields, push buttons, check boxes, radio buttons, list boxes, and combo boxes—cataloging the properties and methods for getting and setting values of the field, and for changing the appearance of the field. That having been done, we now turn to another external processing topic, that of processing collections of fields.

In this our last blog in the series “External Processing of a Field,” we introduce the techniques for working with groups of fields. These techniques may be useful just before submitting data to a server-side script. Before submitting, the data may have to be massaged, put in a format acceptable or expected by the server-side script.

Several years ago I was asked to create a PDF version of a survey for a large tech convention.¹ The server-side script for this survey expected the data to be a single stream of tab-delimited values, in this way, the results of the survey could be brought into a spreadsheet application and analyzed. The approach taken by this survey is a good illustration of processing groups of fields, and is then the theme of this blog. In this blog, we will...

- create a survey form with some required fields,
- check that the required fields are filled in, and
- submit a single stream of tab-delimited data to a server-side script

Server-side Scripts. During the preparation of this article, I wrote two short server-side scripts,

1. `apb_tabsave_fdf.asp`: An ASP file using vbscript and the **PDF Toolkit**.³ The data is submitted as FDF.
2. `apb_tabsave_html.asp`: For this script, the data needs to be submitted as URL - encoded HTML. The **PDF Toolkit** is not used.

The scripts are attached to this blog. The author, that’s me, accepts no liabilities, the script is offered ‘as is,’ no support is provided, so don’t email me. I am of the philosophy, “It worked for me, it should work for you.”

¹If you are interested, there was a **Planet PDF** article about this PDF survey titled “Seybold PDF Survey in PDF also worthy of study,” by Kurt Foss.²

³The PDF Toolkit is available from the [Acrobat Developer Center](#).

11.1. Creating a Survey Form

The survey is simple enough, see the [Short Sample Survey](#) below. I've included in it all form elements except check boxes. (The radio buttons are set to select in unison.) The Name and the Platform questions are required. You don't have to give your true name, just your favorite *pseudo name*.

When the Submit button is pressed, JavaScript executes and gathers the results from the form fields, merges the results into a tab-delimited string, and places this string in the field `dbRecord`. Such a field is normally hidden from view, but in this survey, it is visible, it is the text field in the lower-right corner with the dashed border.

Title: Select one from the list, or enter your own.

Name: * First Name Last Name

Platform: * Windows Mac OS Linux

Application: What are the PDF applications you use? Multiple selections are allowed.

Comments:

Short Sample Survey

Shift-Click the Submit Button: Before you submit, fill in the survey and click the Submit button with the Shift key pressed. The JavaScript verifies the required fields are filled in, gathers the data, and places it in the `dbRecord` field for your viewing pleasure. (See the lower-right corner of the survey.) The data is not submitted, however.

Click the Submit Button: Pressing the Submit button—without pressing the Shift key—submits the data. A limited number of submits is permitted (though I won't say how many). When the Submit button is clicked, a check of the required fields is made, the data is gathered and placed in `dbRecord`, and the single field `dbRecord` is submitted to the server-side script `apb_tabsave_fdf.asp`.

11.2. Check Required Fields

Normally when you submit data, Acrobat/AR⁴ checks which fields being submitted are required. If one or more required fields are not filled, the submit operation is terminated, and an alert box appears to notify the user that a required field is not filled in.

In our situation, the required fields are not being submitted (only the `dbRecord` field is submitted), so Acrobat/AR gives no notice; therefore, *we must check the required fields ourselves*.

Reset all fields with the `Reset` button. Now press Submit while holding down the shift key to see the checking of the required fields in action. Fill in the fields as required.

Title:
Select one from the list, or enter your own.

Name:*
First Name Last Name

Platform:*
Windows Mac OS Linux

Application:
What are the PDF applications you use? Multiple selections are allowed.

Comments:

Short Sample Survey: FDF Submit

When you shift-click the Submit button, the document function `checkRequiredFields()` is executed. If that function returns `true`, the document function `gatherSurveyData()` is executed. This function is the one that populates the `dbRecord` field.

Below is a verbatim listing of the function `checkRequiredFields()` that is called to check required fields. Extensive commenting is shown in red.

⁴AR = Adobe Reader

An important method for dealing with collections of fields is `Doc.getNthFieldName`, which retrieves the name of the N^{th} field in the document. The total number of fields can be acquired from the property `Doc.numFields`. A field property used in the code below is `Field.type`, it returns the type of field button, text, listbox, combobox, checkbox, and radiobutton. This allows you to customize the extraction of information, as a function of the type of field.

```
function checkRequiredFields()
{
    // Use Doc.numFields to get the number of fields in the document
    var n = this.numFields;
    for (var i=0; i<n; i++) {
        // Use Doc.getNthFieldName() to loop through all fields in this document.
        // The method returns the terminal name of the field. Thus, Name.First
        // and Name.Last are return separately.
        var fname=this.getNthFieldName(i);
        // Get the field object for the current field name fname.
        var f=this.getField(fname);
        // If this field is not a button and is required, we process it.
        // Note: a button does not have a required property, so we must
        //      avoid querying this property for a button, otherwise an
        //      exception is thrown.
        if ( (f.type != "button") && f.required ) {
            // If this field is a text field, we remove any white space.
            var value = ( f.type=="text") ? f.value.replace(/\s*/g,"") : f.value;
            // If the value is the same as the default value, the user
            // has not touched this field
            if ( value == f.defaultValue ) {
                // we maintain a global variable lastRequiredField which contains
                // of the last required processed. We make that border black.
                if ( lastRequiredField !="" ) {
                    var g=this.getField(lastRequiredField);
                    g.strokeColor=color.black;
                }
                // Set the current field name to lastRequiredField.
                lastRequiredField=fname;
                // Set the focus on this latest field, set the stroke color
                // to red, and alert the user with a custom message.
                f.setFocus();
                f.strokeColor=color.red;
                app.alert({cMsg: aRequiredMsg[fname],
                    cTitle: "AcroTeX PDF Blog Alert"});
                // If a required field is not filled, we return false
                return false;
            }
        }
    }
}
```

```

// If we did not return earlier, everything is OK, so we set the stroke color
// to black for the lastRequiredField.
if ( lastRequiredField != "" ) {
    var g=this.getField(lastRequiredField);
    g.strokeColor=color.black;
    lastRequiredField = "";
}
return true;
}

```

After `checkRequiredFields()` return true, `gatherSurveyData()` is then executed. In this code, we use a different approach. The fields we want to process are listed in an array, `aFields`. We loop through this array using a `for` loop, again, we use `Field`.type to specialize our analysis of the field. One other note, if `f` is the current field, we set `g=f.getFieldArray()`. The `Field.getFieldArray()` method returns an array of children fields, in the case of hierarchical naming (and we have that for the `Name.First` and `Name.Last` fields). If `g.length` is 1, then we assume the field `f` is a field with a terminal name (no children).

```

// We list the fields of this survey in an array.
var aFields = new Array("title", "Name", "platform", "appl", "comments");
function gatherSurveyData()
{
    // cFormData will hold the tab delimited string
    var cFormData="", f,g;
    var dbRecord = this.getField("dbRecord");
    for ( var i=0; i < aFields.length; i++) {
        // We go through the fields listed in the aFields array
        f = this.getField(aFields[i]);
        // If the length of g is 1, we have a terminal name
        g = f.getFieldArray();
        if ( g.length == 1 ) {
            // We are processing a field with a terminal name
            switch(f.type) {
                // If its a list box we must be sure to have write exactly
                // f.numItems tab fields
                case "listbox":
                    var n=f.numItems;
                    // array a will have exactly f.numItem tabs
                    var a = new Array();
                    for ( var k=0; k<n; k++) a.push("\t");
                    if ( typeof f.currentValueIndices == "number" )
                        // User has selected only one item from list box
                        // Enter choice into appropriate location in a
                        a[f.currentValueIndices]="\t"
                            +f.getItemAt(f.currentValueIndices,true));
                    else {

```

```

        // User has selected multiple items from list box
        for ( var k=0; k<f.currentValueIndices.length; k++) {
            // Enter choice into appropriate location in a
            if ( f.getItemAt(f.currentValueIndices[k],true)!="NR")
                a[f.currentValueIndices[k]]=("\t"
                    +f.getItemAt(f.currentValueIndices[k],true));
        }
    }
    // we join the matrix, and concatenate it to cFormData
    cFormData += (a.join(""));
    break;
// if radio button, and value is "Off", write "NR" instead
// "NR" for no response
case "radiobutton":
    cFormData += ("\t"+ ((f.value!="Off") ? f.value : "NR"));
    break;
default:
    // If not one of the previous types, we store the
    // value in cFormData, adding in a tab
    cFormData += ("\t"+f.value);
}
} else {
    // The field is not a terminal field, it has children
    // The array g=f.getArray() holds the child objects
    switch(g[0].type) {
        // If a text field, we add in their values to cFormData
        case "text":
            for ( var j=0; j<g.length; j++)
                cFormData += ("\t"+g[j].value);
            break;
        }
    }
}
// Finally, we set the value of ebRecord to cFormData.
dbRecord.value=cFormData;
return true;
}

```

11.3. Submit a Single Stream

Finally, if you haven't done so already, fill in the form field and submit. I've put a limit on the number of submits. Otherwise, I fear, you would sit there and hit submit all day. But, PDFers are generally responsible, so no worries.

Title:	Select one from the list, or enter your own.	
Name:*	First Name	Last Name
Platform:*	Windows Mac OS Linux	
Application:	What are the PDF applications you use? Multiple selections are allowed.	
Comments:		

Short Sample Survey: FDF Submit

The Submit button calls a document level function `submitThisSurvey()`, the verbatim code follows.

```
function submitThisSurvey() {
    // check for required fields
    var retn = checkRequiredFields();
    // do nothing if retn=false, otherwise...
    if ( retn ) {
        // gather data
        var retn = gatherSurveyData();
        if ( retn ) {
            // if user is not pressing the shift key, submit
            if ( !event.shift ) {
                if ( submitThisSurvey.count < 4 ) {
                    this.submitForm({
                        cURL: "http://localhost/scripts/apb_tabsave_fdf.asp#FDF",
                        aFields: "dbRecord",
                        cSubmitAs: "FDF"
                    });
                    // increment the count
                    submitThisSurvey.count++;
                }
            }
        }
    }
}
```

```
        } else
            // Admonish the user who presses Submit too many times
            app.alert("Only 4 submissions are permitted. "
                +"Please find some other activity to occupy your time.");
        }
    }
}
// When doc is loaded, set count to zero
submitThisSurvey.count=0;
```

Hope it worked for you, it worked for me.

11.4. Submit as HTML

We can do the same thing with an HTML submit; however, we cannot remain in the PDF, we must go off to an HTML page. The server-side script is `apb_tabsave_html.asp`.

Title:	Select one from the list, or enter your own.	
Name:*	First Name	Last Name
Platform:*	Windows	Mac OS Linux
Application:	What are the PDF applications you use? Multiple selections are allowed.	
Comments:		

Short Sample Survey: HTML Submit

HTML Submit. The code for the Submit button is identical to the FDF submit, except for the `cURL` and `cSubmitAs` parameters. We specify `apb_tabsave_html.asp` in the `cURL` parameter.

```
function submitThisSurveyHTML() {
    // check for required fields
    var retn = checkRequiredFields();
    // do nothing if retn=false, otherwise...
    if ( retn ) {
        // gather data
        var retn = gatherSurveyData();
        if ( retn ) {
            // if user is not pressing the shift key, submit
            if ( !event.shift ) {
                if ( submitThisSurvey.count < 4 ) {
                    this.submitForm({
                        cURL: "http://localhost/scripts/apb_tabsave_html.asp",
                        aFields: "dbRecord",
                        cSubmitAs: "HTML"
                    });
                    // increment the count
                    submitThisSurvey.count++;
                } else
                // Admonish the user who presses Submit too many times
                app.alert("Only 4 submissions are permitted. "
                    +"Please find some other activity to occupy your time.");
            }
        }
    }
}
```

Well, that's pretty much it for now, I simply must get back to my retirement. ☹