AcroTEX.Net

AeB Pro
A Note on JS Helper Commands

D. P. Story

© 2016 dpstory@acrotex.net
AeB Blog September 14, 2016

http://www.acrotex.net
Published: September 14, 2016

Table of Contents

1. Introduction

AeB Pro (ctan.org/pkg/aeb-pro), dated 2016/08/03 or later, defines what are called 'JavaScript Helper commands' to simplify the use of security restricted JavaScript methods within the `docassembly` environment. At the time of this writing, the list of these JS helper commands is as follows:

```
\addWatermarkFromFile   \importIcon     \importSound
\importDataObject       \appopenDoc     \docSaveAs
\createTemplate         \executeSave    \insertPages
\extractPages           \mailDoc
```

Each of these expands to a JavaScript function, with the exception of `\executeSave`, all commands have an argument of the form:

```
⟨\cmd⟩({⟨KV-pairs⟩})
```

where *KV-pairs* use the syntax of the form ⟨*key*⟩: ⟨*value*⟩; keys-value pairs are separated by a comma (,). The helper functions are documented in more detail in the AeB Pro reference manual this is distributed with AeB Pro.

Normally, the target document of the helper commands is the current document, but this can be changed, as will be illustrated in this article.

2. The JS Helper commands

In this article, we focus on `\docSaveAs` and develop techniques and examples for changing the target document. The JS Helper command `\docSaveAs({ cPath: myPath })`, for example, expands to

```
aebTrustedFunctions(\theDocObject,aebDocSaveAs,{ cPath: myPath })
```

where `\theDocObject` expands to the default Doc object `this` and `aebDocSaveAs` is a 'trusted' version of the JavaScript method. *Doc*`.saveAs()`. Since `\theDocObject` expands to 'this', in actuality the above display appears as,

```
aebTrustedFunctions(this,aebDocSaveAs,{ cPath: myPath })
```

The trusted functions `aebTrustedFunctions()` and `aebDocSaveAs()`, which are defined in `aeb_pro.js`, are ultimately executed as,

```
app.beginPriv();
var retn = this.saveAs({ cPath: myPath });
app.endPriv();
```

where the `this.saveAs()` method is executed in a raised privileged context, from within the `app.beginPriv()`/`app.endPriv()` pair. Details about trusted functions, privilege, and other JavaScript API for Acrobat may be found in [1]. In the display above, `this.saveAs()`, the '`this`', refers to saving the current document.

The following example demonstrates how to use the `\docSaveAs` under its default assumptions that `\theDocObject` expands to '`this`'.

```
 1  \begin{docassembly}
 2  var filename=this.documentFileName;
 3  var pos=filename.lastIndexOf(".pdf");
 4  var basename=filename.substring(0, pos);
 5  // Save under a different name
 6  console.println(basename+"_tmp.pdf");
 7  \docSaveAs({cPath: basename+"_tmp.pdf"});
 8  // Now save same document as a PNG file
 9  console.println(basename+"_tmp.png");
10  \docSaveAs({cPath: basename+"_tmp.png",
11      cConvID: "com.adobe.acrobat.png"});
12  \end{docassembly}
```

This example is attached to this blog article under the name of docSaveAs1.tex. The gist of the docassembly code is that the base name of the current document is obtained in lines (1)–(4); a copy of the document is saved in line (7); and the PDF is converted into a series of PNG images in lines (10)–(11). Cool!

Saving a different document. Saving or converting a different document, other than the current one, requires some new techniques. The problem is how to change the definition of \theDocObject from 'this' to some other Doc object?

The following example demonstrates how to change \theDocObject to some other Doc object in the JavaScript helper command \docSaveAs. The code below is reproduced from docSaveAs2.tex, which is attached to this PDF. The example does some 'amazing' things, as explained after the code listing.

```
 1  \chngDocObjectTo{\oDoc}{oDoc}
 2  \declareImageAndPlacement{name=dpsIcon,
 3      path=dpsweb-captioned.pdf, placement=dpsImage}
 4  \begin{docassembly}
 5  var oDoc=this.openDataObject("attach1");
 6  var f = oDoc.getField("Message");
 7  f.value = "Hello! Welcome to my world, the AeB Blog site! "
 8      + "I am glad you joined me. "
 9      + "I do hope your interest in these topics continues. dps";
10  oDoc.flattenPages();
11  var _path=this.path;
12  var pos=_path.lastIndexOf("/");
13  _path=_path.substring(0,pos)+"/dpsweb-captioned.pdf";
14  \docSaveAs\oDoc({ cPath: _path });
15  oDoc.closeDoc();
16  \insertPreDocAssembly
17  this.removeDataObject("attach1");
18  \executeSave();
19  \end{docassembly}
20
21  \begin{document}
22  A message from the author:
23      \raisebox{-1in}{\placeImage{dpsImage}{2in}{2in}}
24  \end{document}
```

The document itself is very short, lines (21)–(24), but has a text field in the form of `\placeImage`, the field name of which is `dpsImage`. The `docassembly` code targets that field.

In line (5), we obtain a new `Doc` object by the name of `oDoc`, this is the `Doc` object of the attachment named `attach1`. Anticipating that, in line (1), the (new) command `\chngDocObjectto` is used to declare `\chngDocObjectTo{\oDoc}{oDoc}`; this associates the command `\oDoc` (or any command name of your choosing) with `oDoc`.

In line (2), we declare that the file (`dpsweb-captioned.pdf`) is to be placed in the button field `dpsImage`, given in line (23). The file `dpsweb-captioned.pdf` will be dynamically created by the rest of the `docassembly` code. Follow me so far?

The code performs the following tasks:

1. In line (5), it get the `Doc` object of the attached file (referenced by by `attach1`) but whose filename is `dpsweb.pdf`.

2. The file `dpsweb.pdf` has a text field overlaid by the name of `Message`. In lines (6)–(9), we get that field object and using it, write some text to that field.

3. We flatten the field in line (10). The reason we do this is the form field does not survive the the `\docSaveAs` step in line (14), otherwise.

4. In lines (11)–(13), we get the path to the current document, remove the file name (`dpsweb.pdf`) and replace it with `dpsweb-captioned.pdf`.

5. Now, this is the step where we apply `\docSaveAs`, but not in its ‘default’ settings. The code in line (14) reads `\docSaveAs\oDoc({ cPath: _path });`, where the command `\oDoc` immediately follows `\docSaveAs`, followed by the ‘usual’ argument of a JavaScript helper file. In this case, line (14) save the document corresponding to `oDoc` (as passed by `\oDoc`. The effect is to save `dpsweb.pdf` as `dpsweb-captioned.pdf`, with the caption inserted and flattened.

6. In line (15) we close `oDoc`, for we are done with that `Doc` object.

7. Line (16) inserts the command `\insertPreDocAssembly`, which are the code lines generated by lines (2)-(3). Executing these lines populates the button field `dpsImage` with `dpsweb-captioned.pdf`.

8. Just to be fancy, we remove `dpsweb.pdf` as an attachment in line (17), my formerly favorite number.

9. Finally, we save the current document in line (18).

The same thing can be done using rich text, but that must wait for another day.

Now, back to my retirement. DS

References

[1] "JavaScript for Acrobat API Reference", May 2015, Adobe Systems, Inc., http://adobe.com/devnet/acrobat/documentation.html 3