The docassembly package

D. P. Story
Email: dpstory@acrotex.net

processed June 18, 2021

Contents

1 ⟨∗package⟩

1 Introduction

The docassembly package provides access some security related features of the Acrobat JavaScript API. The content of this package was extracted from aeb_pro so these important and useful methods can be available to document authors that own Acrobat, but enjoy the use of pdflatex, xelatex, lualatex, ps2pdf, or (in the worst case scenario) Adobe Distiller as PDF creators.

> **Important:** Run for your lives! This package requires Adobe Acrobat! Run! Do not look back!. However, if it makes you feel better, you can use any of the usual PDF creators (pdflatex, lualatex, xelatex, dvips -> ps2pdf, or even the much hated dvips -> Adobe Distiller).

This package requires insdljs,[1] which loads hyperref. It also requires the successful installation of the file `aeb_pro.js`, which is distributed by the acrotex-js package. Version 1.6.1 of `aeb_pro.js` is required.[2] Installation of `aeb_pro.js` is explained in `install_jsfiles.pdf`, found in the `docs` folder of the acrotex-js package.

2 Documentation

The documentation of this file is minimal. Most of the "teaching" of how to use the methods supported by this package is contained in the demo files.

The package defines one environment, the `docassembly` environment, and several "helper" commands that are used within the `docassembly` environment. Refer to Section 4 for details.

Installation of `aeb_pro.js`. This file is essential to the correct functioning of this package. It is important, therefore, to correctly install it. Follow the instructions of `docs/install_jsfiles.pdf`.

Demo files. Demo files are contained in the `examples` folder. They illustrate all the security restricted methods of this package. Most of the demo files have a minimal preamble. Several of the files also uses eforms, since form fields are used in those demo files.

End this package early. If aeb_pro is already loaded, terminate this package early since aeb_pro contains the content of this package already.

```
\@ifpackageloaded{aeb_pro}{\PackageInfo{docassembly}
  {aeb_pro detected, early exit from\MessageBreak
   the docassembly package}\endinput}{}
\edef\da@restoreCats{%
    \catcode`\noexpand\"=\the\catcode`\"\relax
    \catcode`\noexpand\,=\the\catcode`\,\relax
```

[1] Part of the acrotex bundle

[2] If you already have aeb_pro installed on your system, be sure you have Version 1.6.1 of `aeb_pro.js`, if not, use the version of `aeb_pro.js` that comes with acrotex-js.

```
    \catcode`\noexpand\(=\the\catcode`\(\relax
    \catcode`\noexpand\!=\the\catcode`\!\relax
}
\@makeother\"\@makeother\,\@makeother\(\@makeother\!
```

3 Package requirements

Require insdljs for this little exercise.

```
\RequirePackage{insdljs}
```

The critical JavaScript files are no longer distributed with this package, but are distributed by the acrotex-js package. This package must be installed, and the instructions in the `docs` folder followed to propery install the critical JavaScript file `aeb_pro.js`.

```
\IfFileExists{acrotex-js.sty}{\let\reqpkg\relax}
```

If the style file `acrotex-js.sty` is detected, we assume the document author has correctly installed `aeb_pro.js`, if not, we require this package, which may be automatically installed on some TEX systems.

```
  {\PackageWarningNoLine{docassembly}
   {The acrotex-js package is required.\MessageBreak
    Before continuing, install this package,\MessageBreak
    read the documentation, and place\MessageBreak
    aeb_pro.js in the expected folder}
    \def\reqpkg{\usepackage{acrotex-js}}
  }
\reqpkg
\execJSOn
```

4 Document Assembly Methods

Special "helper" commands and one environment are defined to take advantage of Acrobat's extensive library of security restricted methods. It is assumed the document author has properly installed `aeb_pro.js`.

4.1 The `docassembly` environment

`docassembly` This is a wrapper environment for the `execJS` environment of the insDLJS Package. Place JavaScript lines in this environment and the script will execute one time after the PDF has been created and opened in Acrobat Pro for the first time.

```
\newenvironment{docassembly}{\execJS{docassembly}}{\endexecJS}
```

4.2 Supported Assembly JS API

These are convenience commands – called JavaScript helper commands – to executing security restricted JavaScript. The JS methods are defined in the

aeb_pro.js file, kept as folder JavaScript. These commands are executed in a verbatim environment where '\' is still the escape character. Each of the JavaScript helper commands expects a left parenthesis '(' following the command name *on the same line* as the command name.[3] See the example below for correct usage.

```
\begin{docassembly}
\addWatermarkFromFile({
    bOnTop:false,
    cDIPath:"/C/AcroTeX/AcroPackages/ManualBGs/Manual_BG_Print_AeB.pdf"
});
\end{docassembly}
```

For each of the methods below, see the *JavaScript for Acrobat API Reference*.

The command \theDocObject is normally set to this, meaning the current document. You may need to set it to some other doc object if you are trying to access a doc object other than the current one. The following are support commands for changing \theDocObject from within the docassembly environment.

\chngDocObjectTo All the JavaScript helper commands use \theDocObject, which is defined to be the this object. To change it within the docassembly environment is difficult. The next command aids in that problem.

```
\let\ap@mrk\@empty
\def\ap@gobtocomma#1,{}
\providecommand\chngDocObjectTo[2]{%
    \def#1##1\ap@mrk{#2,\ap@gobtocomma##1}}
```

The above defines a new command given by #1. The command has one argument which is all content up to the terminating mark \ap@mrk. The trick to removing \thisDocObject and replacing it with #2, in the above definition, we insert '(#2' followed by \ap@gobtocomma, which absorbs \thisDocObject, (absorbs everything through the first comma), followed by all content (##1); the second \@gobble absorbs the left parenthesis that opens the argument.

```
\def\ap@TF{aebTrustedFunctions}
```

An example of usage of \chngDocObject is \chngDocObjectTo{\newDO}{doc} expanded above the docassembly environment. Later, we can say,

```
\chngDocObjectTo{\newDO}{doc}
\begin{docassembly}
...
\docSaveAs\newDO({ cPath: _path });
...
\end{docassembly}
```

That is, it is placed immediately after any of the commands below that uses \theDocObject.

[3] This requirement is consistent with JavaScript function usage.

\theDocObject This command is used in the definition of all JavaScript helper commands, as seen in the definition of \DeclareJSHelper below. It is set to the doc object this. It can be changed using \chngDocObjectTo, as described above.

```
\def\theDocObject{this}
```

\DeclareJSHelper A general purpose command for defining what I am calling JavaScript helper commands.

```
\providecommand\DeclareJSHelper[2]{%
    \def#1##1({\ap@TF(##1\theDocObject,#2,\ap@mrk}}
```

For example, we declare \DeclareJSHelper{\docSaveAs}{aebDocSaveAs} below, the declaration defines a new command, \docSaveAs:

```
\def\docSaveAs#1({\ap@TF(#1\theDocObject,aebDocSaveAs,\ap@mrk}}
```

Note that the argument of \docSaveAs is delimited by the left parenthesis, thus #1 is everything through that opening parenthesis. This approach allows more flexibility in the definition, there can be spaces following the command name \docSaveAs ({path: _path}), for example.

\retnAbsPathAs Several methods require an absolute path to the current folder. The code is,

```
var _path=this.path;
var pos=_path.lastIndexOf("/");
_path=_path.substring(0,pos);
```

We simplify this code for the document author in the form of the command \rtnAbsPathAs(⟨*js-var*⟩);, where ⟨*js-var*⟩ is a JavaScript variable that will hold the absolute path to the current folder; eg, \rtnAbsPathAs(_path); expands to the above code.

```
\def\retnAbsPathAs(#1){var #1=this.path;^^J%
  var pos=#1.lastIndexOf("/");^^J%
  #1=#1.substring(0,pos)}
```

We new begin the documentation of the "helper" commands. For documentation of the arguments of these commands, refer to the *JavaScript™ for Acrobat© API Reference*.

4.2.1 \addWatermarkFromFile

\addWatermarkFromFile This is the method *Doc*.addWatermarkFromFile.
Demo file: watermark-file.tex

```
\DeclareJSHelper{\addWatermarkFromFile}{aebAddWatermarkFromFile}
```

4.2.2 \addWatermarkFromText

\addWatermarkFromText This is the method *Doc*.addWatermarkFromText.
Demo file: watermark-text.tex

```
\DeclareJSHelper{\addWatermarkFromText}{aebAddWatermarkFromText}
```

4.2.3 \importIcon

\importIcon This is the method *Doc*.importIcon.
Demo file: import-icons.tex

```
37 \DeclareJSHelper{\importIcon}{aebImportIcon}
```

4.2.4 \importSound

\importSound This is the method *Doc*.importSound.
Demo file: import-sound.tex

```
38 \DeclareJSHelper{\importSound}{aebImportSound}
```

4.2.5 \appopenDoc

\appopenDoc This is the method app.openDoc. Opens a document and return a Doc object.
Demo file: open-doc.tex

```
39 \DeclareJSHelper{\appopenDoc}{aebAppOpenDoc}
```

4.2.6 \docSaveAs

\docSaveAs This is the method *Doc*.saveAs. Saves a PDF or converts a PDF to another format to a specified path.
Demo file: doc-saveas.tex

```
40 \DeclareJSHelper{\docSaveAs}{aebDocSaveAs}
```

4.2.7 \insertPages

\insertPages This is the method *Doc*.insertPages.
Demo file: insert-pages.tex

```
41 \DeclareJSHelper{\insertPages}{aebInsertPages}
```

4.2.8 \extractPages

\extractPages This is the method *Doc*.extractPages.
Demo file: extract-pages.tex

```
42 \DeclareJSHelper{\extractPages}{aebExtractPages}
```

4.2.9 \createTemplate

\createTemplate This is the method *Doc*.createTemplate. This is a feature that I had great hopes for. With templates, you can create hidden pages that can be made visible (in AA); once a template is created, it can be spawned and deleted in AR.

```
43 \DeclareJSHelper{\createTemplate}{aebCreateTemplate}
```

4.2.10 \importDataObject

\importDataObject
\attachFile

This is the method *Doc*.importDataObject, used to attach files to a PDF; alternately, \attachFile is a more intuitive name for the operation performed.
Demo file: attach-files.tex

```
\DeclareJSHelper{\importDataObject}{aebImportDataObject}
\DeclareJSHelper{\attachFile}{aebImportDataObject}
```

4.2.11 \executeSave

\executeSave

To save the document, use at the *end of the doc assembly environment*. Usage: \executeSave(). The \@gobble used below absorbs the comma that is placed immediately after the second argument by \DeclareJSHelper.

```
\DeclareJSHelper{\executeSave}{aebSaveAs,"Save"\@gobble}
```

4.2.12 \mailDoc

\mailDoc

This is the method *Doc*.mailDoc. Attach the current PDF and email it someone. Must have a default mail client registered by Acrobat under Edit > Preferences > Mail Accounts.
Demo file: mail-doc.tex

```
\DeclareJSHelper{\mailDoc}{aebMailDoc}
```

4.2.13 Signature related commands

Demo files: sign.tex, certifyinvisible.tex

The \sigInfo command is used for entering signing formation into what will become an object. \signatureSign takes no arguments, but uses the info entered by \sigInfo. An example is

```
\begin{docassembly}
\sigInfo{
    cSigFieldName: "sigOfDPS",
    ohandler: security.PPKLiteHandler,
    cert: "<name>.pfx", password: "<password>",
    oInfo: { location: "Niceville, FL",
      reason: "I am approving this document",
      contactInfo: "dpstory@acrotex.net",
      appearance: "My Signature" }
};
\signatureSign
\end{docassembly}
```

\sigInfo

The \sigInfo command is a latex interface to creating the oSigInfo object.

```
\newcommand{\sigInfo}{var oSigInfo=}
\def\sigFieldObj(#1){var oSigField=this.getField(#1)}
```

For the \signatureSetSeedValue, the field object is required. This function assumes that the JavaScript variable oSigField is the field object. For examle,

```
\begin{docassembly}
\sigFieldObj("sigOfDPS");
\signatureSetSeedValue({
    lockDocument:true,
    appearanceFilter:"My Signature",
    reasons: ["This is a reason", "This is a better reason"],
    flags:0x80|0x100|0x8
});
\end{docassembly}
```

The signatureSetSeedValue() method seeds a signature field with various default values available to the signer.

```
\begin{docassembly}
var sv={
    mdp: "defaultAndComments",
    reasons: ["This is a reason", "This is a better reason"],
    flags:0x80|0x100|0x8
};
\sigFieldObj("sigOfDPS");
\signatureSetSeedValue(sv);
\end{docassembly}
```

\signatureSetSeedValue This is the *Field*.signatureSetSeedValue method. The field name is passed to this method through the cSigFieldName property of the oSigField object.

```
\def\signatureSetSeedValue#1{%
    \ap@TF( oSigField, aebSignatureSetSeedValue, }
```

\signatureSign This is the *Field*.signatureSign method. The field name is passed to this method through the cSigFieldName property of the oSigField object. The function \signatureSign takes the info in the oSigInfo object, gets the security handler object, logs into the handler, calls signatureSetSeedValue if the sv property is in the oSigInfo object, and signs the field.

```
\begin{defineJS}[\makecmt\%\dfnJSCR{^^J}]{\signatureSign}
if ( typeof oSigInfo.oHandler=="undefined" )
  oSigInfo.oHandler=security.PPKLiteHandler;
var engine=aebTrustedFunctions( security,%
aebSecurityGetHandler, oSigInfo.oHandler );
var path2Cert = (typeof oSigInfo.path2Cert == "undefined") ? %
aebTrustedFunctions( this, aebAppGetPath,%
{cCategory:"user"} )+"/Security"+"/"+oSigInfo.cert : %
oSigInfo.path2Cert;
aebTrustedFunctions( engine, aebSecurityHandlerLogin,%
{ cPassword: oSigInfo.password, cDIPath: path2Cert});
var oSigField = this.getField(oSigInfo.cSigFieldName);
oSigInfo.oInfo.password=oSigInfo.password;
if ( typeof oSigInfo.sv!="undefined" ) {
```

```
  for (var o in oSigInfo.sv )
    oSigInfo.oInfo[o]=oSigInfo.sv[o];
}
var oSigArgs={ oSig: engine, oInfo: oSigInfo.oInfo };
if ( typeof oSigInfo.cLegalAttest!="undefined" )
  oSigArgs.cLegalAttest=oSigInfo.cLegalAttest;
if ( typeof oSigInfo.cDIPath!="undefined")
  oSigArgs.cDIPath=oSigInfo.cDIPath;
if ( typeof oSigInfo.bUI!="undefined")
  oSigArgs.bUI=oSigInfo.bUI;
aebTrustedFunctions( oSigField, aebSignatureSign, oSigArgs );
\end{defineJS}
```

\certifyInvisibleSign

This is the *Doc*.certifyInvisibleSign method. This command uses the trusted version of certifyInvisibleSign to sign. The command requires that \sigInfo is populated appropriately.

```
\begin{docassembly}
\sigInfo{
    cert: "<name>.pfx",
    password: "<password>",
    cLegalAttest: "Certified using JavaScript",
    bUI:false,
    oInfo: {
      location: "Niceville, FL",
      reason: "I am certifying this document",
      mdp: "defaultAndComments",
    }
};
\certifyInvisibleSign
\end{docassembly}
```

```
\begin{defineJS}[\makecmt\%\dfnJSCR{^^J}]{\certifyInvisibleSign}
if ( typeof oSigInfo.oHandler=="undefined" )
  oSigInfo.oHandler=security.PPKLiteHandler;
var engine=aebTrustedFunctions( security, %
aebSecurityGetHandler, oSigInfo.oHandler );
var path2Cert=aebTrustedFunctions( this, aebAppGetPath, %
{cCategory:"user"} )+"/Security"+"/"+oSigInfo.cert;
aebTrustedFunctions( engine, aebSecurityHandlerLogin, %
{ cPassword: oSigInfo.password, cDIPath: path2Cert});
oSigInfo.oInfo.password=oSigInfo.password;
var oSigArgs={ oSig: engine, oInfo: oSigInfo.oInfo };
if ( typeof oSigInfo.cLegalAttest!="undefined" )
  oSigArgs.cLegalAttest=oSigInfo.cLegalAttest;
if ( typeof oSigInfo.cDIPath!="undefined")
  oSigArgs.cDIPath=oSigInfo.cDIPath;
if ( typeof oSigInfo.bUI!="undefined")
  oSigArgs.bUI=oSigInfo.bUI;
aebTrustedFunctions( this, aebCertifyInvisibleSign, oSigArgs );
\end{defineJS}
```

```
\da@restoreCats
⟨/package⟩
```

5 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

6 Change History

v1.0 (2021/06/11)

General: First publication version of this package 2

v1.1 (2021/06/18)

General: The aeb_pro.js file is now distributed by the acrotex-js package 2