AcroTEX.Net

The AeB Pro Package
Doc/Page Events & Fullscreen Support

D. P. Story

Prepared: August 3, 2016

http://www.acrotex.net
Published: January 12, 2007

Table of Contents

1. Document Actions

In this section we outline the various commands and environments for creating document and page actions for a PDF document.

As you read through this section, keep the console window open to see the various page events reporting back to the console.

1.1. Document Level JavaScripts

Creating document level JavaScript has been part of AeB for many years, use the `insDLJS` environment, as documented in `webeqman.pdf`.

1.2. Set Document Actions

The AeB Pro provides environments for the events willClose, willSave, didSave, willPrint and didPrint JavaScript events. Corresponding LaTeX environments are created: `willClose`, `willSave`, `didSave`, `willPrint` and `didPrint`. These are illustrated in the preamble of this document.

1.3. Open/Close Page Actions

When a page opens or closes a JavaScript occurs. Predefined JavaScript can execute in reaction to these events. AeB Pro provides several commands and environments.

• Open/Close Page Actions for First Page

Because of the way AeB was originally written—exerquiz, actually—, the first page is a special case.

There is a command, `\OpenAction`, that is part of the `insdljs` package for several years, that is used to introduce open page actions:

```
\OpenAction{\JS{%
    console.show();\r
    console.clear();\r
    console.println("Show the output of the page actions");
}}
```

This command goes in the preamble to define action for the first page. This command is capable of defining non-JavaScript action, see the documentation of `insdljs` for some details.

Environments, defined in AeB Pro, but uses macros from `insdljs` are `addJSToPageOpen` and `addJSToPageClose`. When placed in the preamble, these provide JavaScript support for page open/close events of the first page. In the preamble of this document, you'll find

```
\begin{addJSToPageOpen}
var str = "This should be the first page"
console.println(str + ": page " + (this.pageNum+1));
\end{addJSToPageOpen}
```

and

```
\begin{addJSToPageClose}
var str = "This is the close action for the first page!"
console.println(str + ": page " + (this.pageNum+1));
\end{addJSToPageClose}
```

• Open/Close Page Actions for the other Pages

The same two environments `addJSToPageOpen` and `addJSToPageClose` can be used in the body of the text to generate open or close actions for the page on which they appear. It's a rather hit or miss proposition because the tex compiler may break the page at an unexpected location and the environments are processed on the page following the one you wanted them to appear on.

Just below this paragraph are `addJSToPageOpen` and `addJSToPageClose` environments. Will the effects defined by these environments appear on this page or the next?

Another approach to trying to place `addJSToPageOpen` or `addJSToPageClose` on the page you want is to use the `addJSToPageOpenAt` or `addJSToPageCloseAt` environments. These are the same of their cousins, but are more powerful. Each of these takes an argument that specifies the page, pages, and/or page ranges of the open/close effects you want. These two commands can go in the preamble, but I recommend putting them just after the `\begin{document}` and before `\maketitle`, as illustrated in this document.

The two environments take a comma-delimited list of pages and page ranges, for example, an argument might be `{2-6,9,12,15-}`. This argument states that the open or close JavaScript listed in the environment should execute on pages 2 through 6, page 9, page 11, and pages 15 through the end of the document. Very cool!

This is all well and good if you know exactly which pages are the ones you want the effects to appear. What's even more cool is that you can use LaTeX's cross-referencing mechanism to specify the pages. By placing these environments after `\begin{document}`, the cross referencing information (the `.aux`) has been input and you can use `\atPage`, a

special simplified version of \pageref, to reference the pages. See the verbatim listing below.

```
\begin{addJSToPageOpenAt}{1,\atPage{test}-\atPage{exam}}
var str = "Add to open page at pages between \\\\atPage{test} and \\\\atPage{exam} "
    + (this.pageNum+1);
console.println(str);
\end{addJSToPageOpenAt}
```

In the above, we specify a range \atPage{test}-\atPage{exam}, which when expanded becomes a range of 9-14. If the first page number is larger than the second number, the two numbers are switched; consequently, \atPage{exam}-\atPage{test} yields the same results.

```
\begin{addJSToPageCloseAt}{5-8,12,15-}
var str = "Add to close page at page " + (this.pageNum+1);
console.println(str);
\end{addJSToPageCloseAt}
```

Notice that in the addJSToPageOpenAt environment above, page 1 was specified. This specification is ignored. You do remember that the first page events need to be defined in the preamble, don't you.

• Every Page Open/Close Events

As an additional feature, there may be an occasion where you want to define an event for every page. These are handled separately from the earlier mentioned open/closed events so one does not overwrite the other. These environments are everyPageOpen and everyPageClose. They can go in the preamble, or anywhere. They will take effect on the

page they are processed on. Using these environments a second time overwrites any earlier definition. To cancel out the every page action you can use `\canceleveryPageOpen` and `\canceleveryPageClose`. The environments that appear in the preamble are

```
\begin{everyPageOpen}
var str = "every page open";
console.println(str + ": page " + (this.pageNum+1));
\end{everyPageOpen}

\begin{everyPageClose}
var str = "every page close";
console.println(str + ": page " + (this.pageNum+1));
\end{everyPageClose}
```

2. Fullscreen Control

In this section we present the controlling commands for default fullscreen mode and for defining page transition effects.

2.1. `\setDefaultFS`

Set the default fullscreen behavior of Adobe Reader/Acrobat by using `\setDefaultFS` in the preamble. This command takes a number of arguments using `xkeyval`, each key correspond to a JavaScript property of the fullscreen object.

In the preamble of this document, I have placed `\setDefaultFS` specifying that the document should go into fullscreen mode with a random transition for its default transition effect.

```
\setDefaultFS
{%
    fullscreen,
    cursor=delay,
    Trans=Random,
    loop,
    escape
}
```

See the AeB Pro documentation for full documentation on these properties.

2.2. Page Transition Effects

There are two commands `\setPageTransition` and `\setPageTransitionAt`. The former sets the transition effects for the page on which it is processed. It suffers from the same malady as do `addJSToPageOpen` and `addJSToPageClose`, you have to hit the page you want. The latter command is the same remedy, as illustrated below.

```
\setPageTransitionAt{1,\atPage{test}-\atPage{exam},7}{Trans=Blend,PageDur=20,TransDur=5}
```

See the AeB Pro documentation for full documentation on these properties.

3. Test

Hi world! Page 9

Again, hi! Page 10

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Canceling every open page

4. Exam

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Resetting every page open this page

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!

Introducing the AeB Pro Family!