AcroTEX.Net

AcroTEX eDucation Bundle Professional

Enhanced AeB Features using Acrobat Pro

D. P. Story

Prepared: May 9, 2007

www.acrotex.net
Version 1.0

Table of Contents

1. Overview

AeB Pro, package file base name `aeb_pro`, is an assortment of features (see Section 1.2 below) implemented through a combination of pdfmarks, which are native to a PostScript file, and JavaScript techniques, some of which require Acrobat Professional. These features were meant to be used with AeB (AcroTeX eDucation Bundle); in particular, the `insdljs` and `eforms` packages are essential to AeB Pro. The document author must use distiller to create the PDF and use Acrobat Pro 7.0 or later to access the advance JavaScript methods. For the most part, once the document is assembled, it can be viewed by Adobe Reader 7.0 or later.

1.1. Dedication

This is a package that I've been meaning to write for some time, it has had to wait for my retirement. The AeB Pro package includes several techniques that I've developed over the years for my personal use, and a few new ones. The techniques require Acrobat Pro 7.0 or later, as well as the Acrobat Distiller.

As a now former educator, I've always preferred the use of Acrobat/distiller over pdftex/Adobe Reader. I recognize the debt I owe to the Y&Y TeX System,[1] and to Acrobat and distiller.[2] These systems have inspired me and have made it easy to develop new ideas. I believe that if I had not used the Windows/Acrobat platform, I would not have developed all the packages and systems that I did.[3]

I dedicate AeB Pro to Y&Y (developer Berthold K. P. Horn) and to Adobe Systems, developer of Acrobat. Since I entered the Internet education business, I've gotten to know Berthold quite well through our email correspondence, and many of the software engineers of the Acrobat software engineering team.[4] Thank you all for your wonderful work.

1.2. Features

As you might discern from the table of contents, this package features:

1. AeB Central Control: A uniform way of handling the packages in the AcroTeX Family of Software.
2. Supports all fields in the Initial View tab of the Document Properties dialog box.
3. Complete support for document level JavaScripts and for document actions.

[1] Sadly, now out of business. Y&Y was a critically important partner in my efforts: its early use of type 1 fonts made it easy to use different fonts; its excellent dviwindo previewer—still unsurpassed by current previewers—was an essential tool in much of what I did, and really fired my imagination.

[2] Though pdftex and dvipdfm are important applications and have their place in the LaTeX to PDF workflow, I found them too limiting and too slow in development. For Acrobat, you have a team of top professional software developers working on the Acrobat/Adobe Reader applications, as opposed to academics working sporadically on a PDF creator. The viability of the applications (pdftex and dvipdfm) ultimately depend on too few individuals.

[3] An Internet colleague once asked me why I didn't switch over to Linux, I responded that if I had done that, we would not know each other. We were brought together by the software development that I did on the Windows/Acrobat platform. Switching would have shut me down from the beginning.

[4] In the year 2000, I took a seven month sabbatical in San José, CA, and worked on the Acrobat software engineering team, for Acrobat 5.0. Good memories from my days with Adobe remain. I made good friends there.

4. Complete support for page actions, both open and close events.
5. Complete support for fullscreen mode.
6. Support for attaching documents, and for linking to and for launching embedded files.
7. Support for creating a PDF Package, new to version 8 of Acrobat.
8. Support for what I call document assembly methods, which I've found to be very useful through the years. (This technique was developed in the year 2000 while I was out in San José.)
9. Support for the use of Optional Content Groups, rollovers and animations.

I anticipate future developments.

1.3. Requirements

The major requirement of this package is **Acrobat 7.0 Professional** or later,[5] to repeat

Acrobat 7.0 Professional or later and accompanying **Distiller**

are required for this package to perform as designed. Once the document is built, however, **Adobe Reader 7.0**, or later, is sufficient to view the document. This is a reasonable restriction since some JavaScript techniques used by this package require Acrobat Pro. Also, layer (OCG), which AeB Pro uses, creation using pdftex and dvipdfm, the two major applications used by most of the TEX-users to produce PDF (along with pdfwrite[6]), has not been developed. Therefore, I assume you are using **Acrobat 7.0 Pro** and the accompanying **Distiller**. This package supports the use of dvips and dvipsone to produce a PostScript file to distill.

The **AeB Pro** requires the `insdljs eforms` packages, both of which are included with the **AcroTEX eDucation Bundle** (AeB) distribution. The use of the Web package is optional, though highly recommended. These are all meant to fit together as a comprehensive and unified family of packages, after all.

Below is a list of other required packages used by the APB:

1. hyperref: The hyperref bundle should be already on your system, it is standard to most LATEX distributions.
2. `xkeyval`: The very excellent package by Hendri Adriaens. This package allows developers to write commands that take a variety of complex optional arguments. You should get the most recent version, at this writing, the latest is v2.5e (2005/11/25) or later.
3. `xcolor`: An amazing color package by Dr. Uwe Kern. This package makes it easy to write commands to dim the color. Get a recent version, at this writing, the latest is v2.08 (2005/11/25).

[5] In the United States and Europe, Adobe offers a significant academic discount on its software, including **Acrobat 7.0 Pro** and now **Acrobat 8 Pro**. Educators should look into the price structure of **Adobe Acrobat** at their institutions; perhaps, their Department or College can supply a financial grant for the purchase of the software.

[6] I know very little of pdfwrite and its capabilities.

4. `truncate`: This package, by Donald Arseneau, is used in the navigation panel to abbreviate the section titles if they are too wide for the panel. This package is distributed with the APB.
5. `comment`: A general purpose package, Victor Eijkhout, for creating environments that can be included in the document or excluded as comments. A very useful package for LaTeX package developers. This package is distributed with the APB.
6. `eso-pic` by Rolf Niepraschk and `everyshi` by Martin Schröder, these are used by Web to create background graphics and graphic overlays.

One of the extremely nice features of **MiKTeX** is that it can automatically download and install any unknown packages onto your hard drive, so getting the AeB Pro up and running is not a problem!

1.4. The AeB Pro Family of Software

Earlier in the year 2006, I published some packages that pre-date AeB Pro, yet I consider to be part of the AeB Pro family. These are

1. The aebXMP Package: A LaTeX package that fills in the advance metadata. Requires Acrobat 8 Professional, and uses E4X, the xml parser that is built into version 8 JavaScript engine.
2. The AcroSort Package: A novelty package for importing an image that has been sliced into rows and columns and randomly rearranged. The JavaScript does a bubble sort on the picture.
3. AeB Slicing batch sequence: This is a batch sequence for Acrobat Pro that takes the image open in Acrobat and slices it into a specified number of rows and columns, and saves the slices to a designated folder.
4. The AcroMemory Package: A LaTeX package that implements two variations of a memory game: (1) a single game board consisting of a number of tiles, each tile has a matching twin, the object is to find all the matching twins; (2) two game boards, both identical except one has been randomly rearranged, the object is the find the matching pieces in each of the two game boards. The AeB Slicing is used to slice the image into a specified number of rows and columns.

These, as well as the AeB Pro distribution itself, are available through the package web site

www.math.uakron.edu/~dpstory/aeb_pro.html

and through my "commercial" web site www.acrotex.net.

1.5. Package Options

Below is a list of all options of the AeB Pro package:

1. `driver`: Permissible values are `dvipsone` and `dvips`. If the nopro option is taken, in which case AeB Pro acts as a AeB Control Central, then `pdftex`, `dvipdfm` and `textures` are also accepted.

2. AeB Package Options: The AeB Pro package recognizes the components of AeB, these are `web`, `exerquiz`, `dljslib`, `eforms`, `insdljs`, `eq2db`, `aebxmp` and `hyperref`. The value of each of these is a list of options you want that package to use. (The hyperref package is not a component of AeB, but it is such an integral part of AeB that it is included.) See Section 2, page 9.
3. `uselayers`: Taking this option brings in code in support of Optional Content Groups, see Section 12, page 37.
4. `nopro`: If this option is taken, then no code that requires Distiller is input. With the `nopro` option, AeB Pro acts as AeB Control Central allowing the document author have a nice interface to input the various components of the AeB package. See Section 2, page 9.
5. `gopro`: Some components of AeB have a pro option, when you use the gopro option of AeB Pro, the `pro` option is passed to all components of AeB Pro that have a `pro` option.
6. `attachsource`: This key has as its value a list of extensions. For each extension listed, the file `\jobname.ext` will be attached to the parent PDF. See Section 7.1, page 23.
7. `attachments`: This key has its value a list of paths to files to be attached to the parent document. See Section 7.2, page 24.
8. `linktoattachments`: Invoking this option causes code for linking to attachments, or for giving attachments descriptions other than the default ones. See Section 9, page 28.
9. `latin1`: A companion option to `linktoattachments`. When this option is used, the set of latin1 unicodes are input and are available to be used in the descriptions of attachments. See 'Notes on the `<description>`' on page 31.
10. `childof`: In a LaTeX child document, use this option to set the path back to the parent document. See Section 9.6, page 35.

1.6. Installation

We outline the method of installing AeB Pro in this section.

• Unzipping `acrotex.zip`

If you don't have an `acrotex` folder already in your LaTeX distribution, unzip `acrotex_pack.zip` on the search path of your LaTeX system. Unzipping will create a `acrotex` folder with the AeB distribution within.

Should you already have an acrotex folder, unzip `acrotex_pack.zip` instead. This zip file contains only the package files, and the new example files.

In the root folder of `acrotex`, bring the file `acrotex.ins` into your editor and latex it (or latex it from the console command line).

Users of **MiKTeX** need to refresh the filename database.

• **Installing** `aeb_pro.js` **and** `aeb.js`

The JavaScript methods used by the `docassemble` environment, see 'Doc Assembly Methods' on page 24, have a security setting in Acrobat; Acrobat requires that that such methods be *trusted methods*. The file `aeb_pro.js` enables you to execute the doc assembly methods described later without Acrobat raising security exception.

The JavaScript file `aeb.js`, comes with AeB, is only needed if you use Acrobat Pro 8.1 or later. Increased security in that version has made it necessary to install a folder JavaScript file to be able to install document level JavaScripts.

Start **Acrobat Pro 7.0** or later, and open the console window Advanced > JavaScript > Debugger (`Ctrl+J`). Copy and paste the following code into the window.

```
app.getPath("user","javascript");
```

Now, this the mouse cursor on the line containing this script, press the `Ctrl+Enter` key. This will execute this JavaScript. This JavaScript method returns the path to where `apb.js` should be placed. For example, on my system, the return string is

```
/C/Documents and Settings/story/
    Application Data/Adobe/Acrobat/7.0/JavaScripts
```

Follow the path to this folder. If the `JavaScripts` folder does not exist, create it. Finally, copy both `aeb.js` and `aeb_pro.js` into this folder. Close **Acrobat**, then start it again and look under the Tools menu to verify the presence of the sub-menu AcroTEX Presentation Bundle.

1.7. Examples

The following is a list of the example files that illustrate and test various features of AeB Pro.

1. `aebpro_ex1.tex`: Illustrates the document and page open/close actions and fullscreen support of AeB Pro.
2. `aebpro_ex2.tex`: Demonstrates the features of the `pro` option of the web package, including enhanced control over the layout of section headings and the title page.
3. `aebpro_ex3.tex`: Highlights the attachments options and the doc assembly methods.
4. `aebpro_ex4.tex`: A discussion of layers, rollovers and animation.
5. `aebpro_ex5.tex`: This file discusses linking to attachments and covers commands `\ahyperref`, `\ahyperlink` and `\ahyperextract`.
6. `aebpro_ex6.tex`: Learn how to create a PDF Package out of your attachments.
7. `aebpro_ex7.tex`: Explore the `\DeclareInitView` command, documentation included in this file.
8. `aebpro_ex8.tex`: Details of how to use unicode to set the initial value(s) of field, or as captions on a button.

See the file aebpro_index_ex.tex for a this listing in a separate file.

Throughout this document, the above exercises are referenced using icons in the left margins. These icons are live hyperlinks to the source file or the PDF. For example, we reference aebpro_ex1 in this paragraph. The example files can be found in the examples sub-folder of the aeb_pro distribution.

2. AeB Control Central

The AeB family of software, LaTeX packages all, are for the most part stand alone; however, usually they are used in combination with each other, at least that is the purpose for which they were originally designed. When several members of family AeB are used, they should be loaded in the optimal order. With AeB Pro, you can now list the members of the AeB family you wish to use, along with their optional parameters you wish to use.

The list of AeB components supported by AeB Pro is web, exerquiz, dljslib, eforms, insdljs, eq2db, aebxmp and hyperref.

Simply listing a component will cause AeB Pro to install that component, with its default optional parameters; by specifying a value—a list of options required—will cause AeB Pro to load the package with the listed options.

Example 1: Below is a representative example of the use of the AeB options of AeB Pro, AeB Control Central!

```
\usepackage[%
    driver=dvipsone,
    web={pro,designv,tight,nodirectory,usesf},
    exerquiz={<optional parameters>},
    ...,
    aebxmp
]{aeb_pro}
```

Yes, yes, I know this is not necessary, you can always load the packages earlier than AeB Pro, but please, humor me.

By default, the code for supporting features that require the use of Distiller and Acrobat Pro are included; there is a nopro option that excludes these features. Use the nopro if you only wish to use the AeB Control Center feature to load the various members of the AcroTeX family. If nopro is used, AeB Pro can be used with pdftex and dvipdfm, for example.

See the new AeB manual for documentation on the pro option of Web. The support document aebpro_ex2 also presents a tutorial on the the pro option.

The support file aebpro_ex2 has a section discussing the AeB Control Central, as well as features of the pro option of Web.

3. Declaring the Initial View

`\DeclareInitView` is a "data structure" for setting the Initial View of the Document Properties dialog box, See Figure 1.

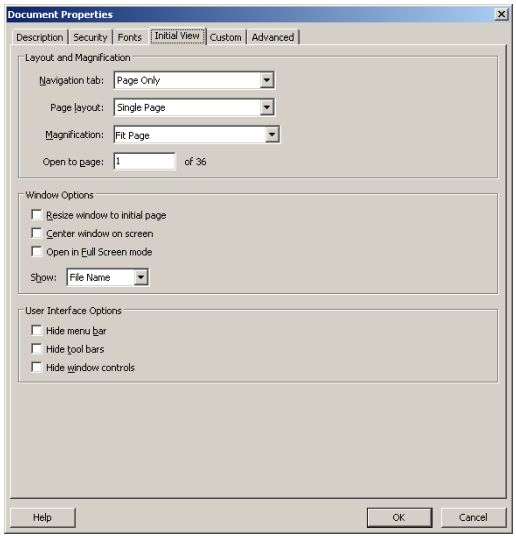

Figure 1: Initial View of Document Properties

`\DeclareInitView` takes up to three key-value pairs, the three keys correspond to the three named regions of the UI (User Interface):

Key	User Interface Name
`layoutmag`	Layout and Magnification
`windowoptions`	Window Options
`uioptions`	User Interface Options

The values of each these three are described in the tables below:

- `layoutmag`: This key sets the initial page layout and magnification of the document. The values of this key are themselves key-values:

Key	Value(s)	Description
navitab	UseNone, UseOutlines, UseThumbs, UseOC, UseAttachments	The UI for these are: Page Only, Bookmarks Panel and Page, Pages Panel and Page, Layers Panel and Page, Attachments Panel and Page, respectively. The default is UseNone
pagelayout	SinglePage, OneColumn, TwoPageLeft, TwoColumnLeft, TwoPageRight, TwoColumnRight	The UI for these are: Single Page, Single Page Continuous, Two-Up (Facing), Two-Up Continuous (Facing), Two-Up (Cover Page), Two-Up Continuous (Cover Page), respectively. The default is user's preferences.
mag	ActualSize, FitPage, FitWidth, FitHeight, FitVisible, or <positive number>	The UI for these are: Actual Size, Fit Page, Fit Width, Fit Height, Fit Visible, respectively. If a positive number is provided, this is interpreted as a magnification percentage. The default is to use user's preferences.
openatpage	<positive number>	The page number (base 1) to open the document at. Default is page 1.

Important: When you set openatpage to a page number other than the first page, be aware that document level JavaScripts are initially imported into the document on the first page. After the file is distilled and the document opens to the page set by openatpage, the document author needs to go to page 1, at which point the document level JavaScripts will be imported. After import, *save the document*, which will save the newly imported JavaScripts with the document.

- windowoptions: The Window Options region of the Initial View tab consists of a series of check boxes which, when checked, modify the initial state of the document window. These are not really Boolean keys. If the key is present, the corresponding box in the UI will be checked, otherwise, the box remains cleared.

Key	Description
fit	Resize window to initial page
center	Center window on screen
fullscreen	Open in Full Screen mode
showtitle	Show document title in the title bar

Note that you can open the document in Full Screen mode using the fullscreen key above, or by using the fullscreen key of the \setDefaultFS. Either will work.

- uioptions: The User Interface Options region of the Initial View tab consists of a series of check boxes which, when checked, hide an UI control. These are not really Boolean keys. If the key is present, the corresponding box in the UI will be checked, otherwise, the box remains cleared.

Key	Description
hidemenubar	Hide menu bar
hidetoolbar	Hide tool bars
hidewindowui	Hide window controls

Important: The hyperref package can set some of these fields of the Initial View tab. The document author is *discouraged* from using hyperref to set any of these fields, though, usually they are overwritten by this package.

Example 2: We set the Initial View tab of the Document Properties dialog box.

```
\DeclareInitView
{%
    layoutmag={mag=ActualSize,navitab=UseOutlines,%
        openatpage=3,pagelayout=TwoPageLeft},
    windowoptions={fit,center,showtitle,fullscreen},
    uioptions={hidetoolbar,hidemenubar,hidewindowui}
}
```

The file aebpro_ex7 is a test file for the features of this section. Use it to explore the properties of the Initial View tab of the Document Properties dialog box.

\DeclareInitView is a companion command to \DeclareDocInfo. Each fills in a separate tab of the Document Properties dialog box. Use the package aebxmp to fill in advance metadata through \DeclareDocInfo.

4. Document Actions

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

4.1. Document Level JavaScripts

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

The document aebpro_ex1 offers an example of the use of the insDLJS environment.

4.2. Set Document Actions

The AeB Pro provides environments for the Acrobat events willClose, willSave, didSave, willPrint and didPrint. Corresponding LaTeX environments are created: willClose, willSave, didSave, willPrint and didPrint.

The example document aebpro_ex1 includes examples of the use of the willClose, willSave, didSave, willPrint and didPrint environments

```
\begin{willClose}
<JS code>
\end{willClose}
```

Environment Description: The JS code in the body of the `willClose` environment will execute just before the document closes.

Environment Location: Place this environment in the preamble.

```
\begin{willSave}
<JS code>
\end{willSave}
```

Environment Description: The JS code in the body of the `willSave` environment will execute just before the document is saved.

Environment Location: Place this environment in the preamble.

```
\begin{didSave}
<JS code>
\end{didSave}
```

Environment Description: The JS code in the body of the `didSave` environment will execute just after the document is saved.

Environment Location: Place this environment in the preamble.

```
\begin{willPrint}
<JS code>
\end{willPrint}
```

Environment Description: The JS code in the body of the `willPrint` environment will execute just before the document is Printed.

Environment Location: Place this environment in the preamble.

```
\begin{didPrint}
<JS code>
\end{didPrint}
```

Environment Description: The JS code in the body of the `didPrint` environment will execute just after the document is Printed.

Environment Location: Place this environment in the preamble.

4.3. Document Open Actions

You can set an action to be performed when the document is open, independently of the page the document is opened at.

```
\addtionalOpenAction{<action>}
```

Command Description: The <action> can be any type of action described in the *PDF Reference*, but it is usually a JavaScript action.

Command Location: Place this command in the preamble.

The following example gets the time the user first opens the document

```
\addtionalOpenAction{\JS{%
    var timestamp = util.printd("mm-dd-yy, H:MM:ss.", new Date());}}
```

Important: This open action takes place rather early in document initialization, before the document level JavaScript is scanned; therefore, the <action> should not reference any document level JavaScript, as at the time of the action, they are still undefined. You are restricted to core JavaScript and the JavaScript API for Acrobat.

Using layers put a natural restriction on the version that can be used to effectively view the document. To put a requirement on the viewer to be used, use the \requiresVersion command.

```
\requiresVersion{<version_number>}
\requiredVersionMsg{<message>}
\alternateDocumentURL{<url>}
\requiredVersionMsgRedirect{<message>}
\afterRequirementPassedJS{<JS code>}
```

Command Location: Place these commands in the preamble.

Command Description: For \requiresVersion, the parameter <version_number> is the minimal version number that this document is made for. If the version number of the viewer is less than <version_number>, an alert box appears, and the document is silently closed, if outside a browser, or redirected, if inside a browser.

Important: The command \requiresVersion needs to be issued *after* any redifinitions of the macros \afterRequirementPassedJS, \requiredVersionMsgRedirect, \requiredVersionMsg and \alternateDocumentURL.

When the document is opened outside a web browser and the version number requirement is not met, the message contained in \requiredVersionMsg appears in an alert box. The default definition is

```
\requiredVersionMsg{%
    This document requires Adobe Reader or Acrobat,
    version \requiredVersionNumber\space or later.
    The document is now closing.}
```

The argument of \requiresVersion is contained in the macro \requireVersionNumber, and this macro should be used in the message, as illustrated above.

When the document is opened in a browser and the version number requirement is not met the message contained in \requiredVersionMsgRedirect appears in an alert box. The default definition is

```
\requiredVersionMsgRedirect{%
    This document requires Adobe Reader or Acrobat,
    version \requiredVersionNumber\space or later.
    Redirecting browser to an alternate page.}
```

The browser is redirected to the URL specified in the argument of \alternateDocumentURL, the default definition of which is

```
\alternateDocumentURL{http://www.acrotex.net/}
```

The command \requiresVersion uses \addtionalOpenAction; if you want to combine several actions, including an action for checking for the version number, use \afterRequirementPassedJS. For example,

```
\afterRequirementPassedJS
{%
    var timestamp = util.printd("mm-dd-yy, H:MM:ss.", new Date());
}
```

The above code will be executed if the version requirement is passed.

You can use \afterRequirementPassedJS, for example, to put deadline to view the document; that is, if the document is opened after a pre-selected date and time, the document should close down (or redirected to an alternate web page).

Important: When using AeB Pro with the uselayers option, the minimum required version is 7. Thus,

```
\requiresVersion{7}
```

should be issued in the preamble of any document that uses layers.

5. 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.

The commands and environments described in this section are illustrated in the support document aebpro_ex1.

5.1. 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{<JS code>}}
```

Command Location: 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.

Below is an example of usage:

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

In addition to `\OpenAction`, `addJSToPageOpen` and `addJSToPageClose` are also defined by AeB Pro. The `<JS code>` is executed each time the page is opened or closed.

```
\begin{addJSToPageOpen}
<JS code>
\end{addJSToPageOpen}
```

For page close events, we have the `addJSToPageClose` environment.

```
\begin{addJSToPageClose}
<JS code>
\end{addJSToPageClose}
```

Environment Description: When placed in the preamble, these provide JavaScript support for page open/close events of the first page.

Below are examples of usage. These appear in the document `aebpro_ex1`.

```
\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}
```

5.2. 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.

```
\begin{addJSToPageOpen}
<JS code>
\end{addJSToPageOpen}
```

```
\begin{addJSToPageClose}
<JS code>
\end{addJSToPageClose}
```

Environment Description: Place on the page that these actions are to apply.

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 as 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.

```
\begin{addJSToPageOpenAt}{<page ranges(s)>}
<JS code>
\end{addJSToPageOpenAt}
```

For page close events, we have the addJSToPageClose environment.

```
\begin{addJSToPageCloseAt}{<page ranges(s)>}
<JS code>
\end{addJSToPageCloseAt}
```

Environment Location: Place these just after \begin{document} and before \maketitle.

Environment Description: When placed in the preamble, these provide JavaScript support for page open/close events of the first page.

Parameter Description: 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. below.

```
\atPage{<label>}
```

Command Description: Returns the page number on which the LaTeX cross-reference label <label> resides.

For example,

```
\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}. If the first page number is larger than the second number, the two numbers are switched; consequently, the specification \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}
```

In the above example, 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.

5.3. 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.

```
\begin{everyPageOpen}
<JS code>
\end{everyPageOpen}
```

For page close events, we have the everyPageClose environment.

```
\begin{everyPageClose}
<JS code>
\end{everyPageClose}
```

Environment Location: Place in the preamble or in the body of the document.

For example,

```
\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}
```

```
\canceleveryPageOpen
\canceleveryPageClose
```

Command Description: Cancels the current everyPageOpen or everyPageClose events. After these commands, additional `everyPageOpen` or `everyPageClose` environments can be used to create different every page events.

6. Fullscreen Support

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

The sample file `aebpro_ex1` demonstrates many of the full screen features described in this section.

6.1. See Fullscreen Defaults: `\setDefaultFS`

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

```
\setDefaultFS{<key-values>}
```

The command for setting how you want to viewer to behave in fullscreen. This command is implemented through JavaScript, as opposed to the `pdfmark` operator. See *JavaScript for Acrobat API Reference* [2], the section on the `FullScreen` object.

Command Location: This command must be executed in the preamble.

Key-Value Pairs: The command has numerous key-value pairs, the defaults of most of these are set in the `Preferences` menu of the viewer. These values are the ones listed in the *Acrobat JavaScript Scripting Reference* [2].

1. `Trans`: permissible values are `NoTransition`, `UncoverLeft`, `UncoverRight`, `UncoverDown`, `UncoverUp`, `UncoverLeftDown`, `UncoverLeftUp`, `UncoverRightDown`, `UncoverRightUp`, `CoverLeft`, `CoverRight`, `CoverDown`,

CoverUp, CoverLeftDown, CoverLeftUp, CoverRightDown, CoverRightUp, PushLeft, PushRight, PushDown, PushUp, PushLeftDown, PushLeftUp, PushRightDown, PushRightUp, FlyInRight, FlyInLeft, FlyInDown, FlyInUp, FlyOutRight, FlyOutLeft, FlyOutDown, FlyOutUp, FlyIn, FlyOut, Blend, Fade, Random, Dissolve, GlitterRight, GlitterDown, GlitterRightDown, BoxIn, BoxOut, BlindsHorizontal, BlindsVertical, SplitHorizontalIn, SplitHorizontalOut, SplitVerticalIn, SplitVerticalOut, WipeLeft, WipeRight, WipeDown, WipeUp, WipeLeftDown, WipeLeftUp, WipeRightDown, WipeRightUp, Replace, ZoomInDown, ZoomInLeft, ZoomInLeftDown, ZoomInLeftUp, ZoomInRight, ZoomInRightDown, ZoomInRightUp, ZoomInUp, ZoomOutDown, ZoomOutLeft, ZoomOutLeftDown, ZoomOutLeftUp, ZoomOutRight, ZoomOutRightDown, ZoomOutRightUp, ZoomOutUp, CombHorizontal, CombVertical. The default is Replace.

The following are new to Acrobat/Adobe Reader version 8: PushLeftDown, PushLeftUp, PushRightDown, PushRightUp, WipeLeftDown, WipeLeftUp, WipeRightDown, WipeRightUp, ZoomInDown, ZoomInLeft, ZoomInLeftDown, ZoomInLeftUp, ZoomInRight, ZoomInRightDown, ZoomInRightUp, ZoomInUp, ZoomOutDown, ZoomOutLeft, ZoomOutLeftDown, ZoomOutLeftUp, ZoomOutRight, ZoomOutRightDown, ZoomOutRightUp, ZoomOutUp, CombHorizontal, CombVertical

The transition chosen by this key will be in effect for each page that does not have a transition effect separately defined for it (by the \setPageTransition command).

2. bgColor: Sets the background color in fullscreen mode. The color specified must be a JavaScript Color array, e.g., bgColor = ["RGB" 0 1 0], or you can use some preset colors, bgColor = color.ltGray.
3. timeDelay: The default number of seconds before the page automatically advances in full screen mode. See useTimer to activate/deactivate automatic page turning.
4. useTimer: A Boolean that determines whether automatic page turning is enabled in full screen mode. Use timeDelay to set the default time interval before proceeding to the next page.
5. loop: A Boolean that determines whether the document will loop around back to the first page.
6. cursor: Determines the behavior of the mouse in full screen mode. Permissible values are hidden, delay (hidden after a short delay) and visible.
7. escape: A Boolean use to determine if the escape key will cause the viewer to leave full screen mode.
8. clickAdv: A Boolean that determines whether a mouse click on the page will cause the page to advance.
9. fullscreen: A Boolean, which if true, causes the viewer to go into full screen mode. Has no effect from within a browser.
10. usePageTiming: A Boolean that determines whether automatic page turning will respect the values specified for individual pages in full screen mode (which can be set through \setDefaultFS.

This example causes the viewer to go into full screen mode, sets the transition to Random, instructs the viewer to loop back around to the first page, and to make the cursor hidden after a short period of inactivity.

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

On closing the document, the user's original full screen preferences are restored.

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.

6.2. Page Transition Effects

The \setDefaultFS command can set the full screen behavior of the viewer for the *entire document*, including a transition effect applicable to all pages in the document; for transition effects of individual pages, use the \setPageTransition command.

```
\setPageTransition{<key-values>}
```

Sets the transition effect for the *next page only*, viewer must be in full screen mode. The command \setPageTransition is implemented using the pdfmark operator.

Command Location: This command should be used in the preamble for the first page, and between slides for subsequent pages.

Key-Value Pairs: The \setPageTransition command has several key-value pairs:

1. Trans: permissible values are NoTransition, UncoverLeft, UncoverRight, UncoverDown, UncoverUp, UncoverLeftDown, UncoverLeftUp, UncoverRightDown, UncoverRightUp, CoverLeft, CoverRight, CoverDown, CoverUp, CoverLeftDown, CoverLeftUp, CoverRightDown, CoverRightUp, PushLeft, PushRight, PushDown, PushUp, PushLeftDown, PushLeftUp, PushRightDown, PushRightUp, FlyInRight, FlyInLeft, FlyInDown, FlyInUp, FlyOutRight, FlyOutLeft, FlyOutDown, FlyOutUp, FlyIn, FlyOut, Blend, Fade, Random, Dissolve, GlitterRight, GlitterDown, GlitterRightDown, BoxIn, BoxOut, BlindsHorizontal, BlindsVertical, SplitHorizontalIn, SplitHorizontalOut, SplitVerticalIn, SplitVerticalOut, WipeLeft, WipeRight, WipeDown, WipeUp, WipeLeftDown, WipeLeftUp, WipeRightDown, WipeRightUp, Replace, ZoomInDown, ZoomInLeft, ZoomInLeftDown, ZoomInLeftUp, ZoomInRight, ZoomInRightDown, ZoomInRightUp, ZoomInUp, ZoomOutDown, ZoomOutLeft, ZoomOutLeftDown, ZoomOutLeftUp, ZoomOutRight, ZoomOutRightDown, ZoomOutRightUp, ZoomOutUp, CombHorizontal, CombVertical. The default is Replace.

   The following are new to Acrobat/Adobe Reader version 8: PushLeftDown, PushLeftUp, PushRightDown, PushRightUp, WipeLeftDown, WipeLeftUp, WipeRightDown, WipeRightUp, ZoomInDown, ZoomInLeft, ZoomInLeftDown, ZoomInLeftUp, ZoomInRight, ZoomInRightDown, ZoomInRightUp, ZoomInUp, ZoomOutDown, ZoomOutLeft, ZoomOutLeftDown, ZoomOutLeftUp,

ZoomOutRight, ZoomOutRightDown, ZoomOutRightUp, ZoomOutUp, CombHorizontal, CombVertical

These values are the ones listed in the *Acrobat JavaScript Scripting Reference* [2].

2. TransDur: Duration of the transition effect, in seconds. Default value: 1.
3. Speed: (APB 2.0) Same as TransDur, the duration of the transition effect, except this key takes values Slow, Medium or Fast, corresponding to the Acrobat UI. If TransDur and Speed are both specified, Speed is used. Use TransDur for finer granularity.
4. PageDur: The *PDF Reference, version 1.6* [5], describes this as "The page's display duration (also called its advance timing): the maximum length of time, in seconds, that the page is displayed during presentations before the viewer application automatically advances to the next page. By default, the viewer does not advance automatically."

For example,

```
\setPageTransition{Trans=Blend,PageDur=20,TransDur=5}
```

The command \setPageTransition suffers from the same malady as do addJSToPageOpen and addJSToPageClose, it has to be placed on the page you want to apply. For this reason, there is the \setPageTransitionAt.

```
\setPageTransitionAt{<page ranges(s)>}{<key-values>}
```

Key-Value Pairs: Same as \setPageTransitionAt

Parameter Description: The parameter <page ranges(s)> has the same format as described in Section 5.2, page 17. This command obeys the \atPage.

For example,

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

7. Attaching Documents

AeB Pro has two options for attaching files to the source PDF. The approach is the importDataObject JavaScript method in conjunction with the FDF techniques.

There are two options for attaching files

1. attachsource is a simplified option for attaching any file of the form \jobname.*ext*.
2. attachments is a general option for attaching a file, as specified by its absolute or relative path.

The file aebpro_ex3 demonstrates many of the commands presented in this section.

7.1. The attachsource option

Use this option to attach a file with the same base name as \jobname.

```
\usepackage[%
    driver=dvips,
    web={
        pro,
        ...
        usesf
    },
    attachsource={tex,dvi,log,tex.log},
    ...
]{aeb_pro}
```

Simply list the extensions you wish to attach to the current document. In the example above, we attach the original source file \jobname.tex, \jobname.dvi, \jobname.log (the distiller log) and \jobname.tex.log (the tex log).

Important: There should be no space following a comma in the lists of extensions. Thus, the list should be

```
attachsource={tex,dvi,log,tex.log}
```

not

```
attachsource={tex, dvi, log, tex.log}
```

or

```
attachsource={tex,
    dvi,
    log,
    tex.log
}
```

However, the following works,

```
attachsource={
    tex,%
    dvi,%
    log,%
    tex.log
}
```

Frankly, the argument list for extensions is so short, there is no reason to put them on separate lines.

One problem with attaching the log file is that the distiller also produces a log file with the same name \jobname.log. Consequently, the log file for the tex file is overwritten by the distiller log file. You'll see from the PDF document, that the log file attached is the one for the distiller.

A work around for this is to latex your file, rename the log file to another extension, such as \jobname.tex.log, then distill. You may want to send that log file so some poor TEXpert for TEXpert analysis!

7.2. The attachments option

The attachments key is for attaching files other than ones associated with the source file. The value of this key is a comma-delimited list (enclosed in braces) of absolute paths and/or relative paths to the file required to attach. For example,

```
\usepackage[%
    driver=dvips,
    web={
        pro,
        ...
        usesf
    },
    attachments={robot man/robot_man.pdf,%
    /C/Documents and Settings/dps/My Documents/birthday17.jpg},
    ...
]{aeb_pro}
```

The first reference is relative to the folder that this source file is contained in (and is attached to this PDF), and second one is an example of an absolute path.

Important: There are some files that Acrobat does not attach, but there is no public list of these. One finds them by discovery, .exe and .zip files, for example.

A trick that I use to send .zip files through the email (they are often stripped away by mail servers) is to *hide* the .zip file in a PDF as an attachment. But since Acrobat does not attach .zip, I change the extension from .zip to .txt, then inform the recipient to save the .txt file and change the extension back to .zip. Swave!

8. Doc Assembly Methods

Ahhhh, document assembly. What can be said? This is a method that I have used for many years and is incorporated into the insdljs package under the name of execJS. Whereas the execJS environment is still available to you, I've simplified things. The term doc assembly refers to the use of the docassembly environment (which is just an execJS environment).

```
\begin{docassembly}
<JS code to be executed when doc is first opened>
\end{docassembly}
```

The execJS/docassembly environments create an FDF file with the various JavaScript commands that were contained in the body of the environment. These environments also place in open page action so that when the PDF is opened for the first time in Acrobat Pro, the FDF file will be imported and the JS will be *executed one time and then discarded*, see [1] for an article on this topic. This technique only works if you have Acrobat Pro.

8.1. Certain Security Restricted JS Methods

In addition to the docassembly environment, AeB Pro also has several macros that expand to JavaScript methods that I find useful. These JavaScript methods are quite useful, yet they have a

security restriction on them; they cannot be executed from within a document, and certainly not by Adobe Reader.

The use of these methods requires the installation of `aeb_pro.js`, the folder level JavaScript file that comes with this package. These methods are normally called from the `docassembly` environment.

```
\addWatermarkFromFile({<key-values>});
```

Command Description: Inserts a watermark into the PDF

Key-Value Pairs: Numerous, see [1]. Here, we mention only two.

1. `cDIPath`: The absolute path to the background or watermark document.
2. `bOnTop`: (optional) A Boolean value specifying the z-ordering of the watermark. If `true` (the default), the watermark is added above all other page content. If `false`, the watermark is added below all other page content.

```
\importIcon({<key-values>});
```

Command Description: Imports icon files[7]

Key-Value Pairs: There are three key-value pairs:

1. `cName`: The name to associate with the icon
2. `cDIPath`: The path to the icon file, it may be absolute or relative
3. `nPage`: The 0-based index of the page in the PDF file to import as an icon. The default is 0.

```
\importSound({<key-values>});
```

Command Description: Imports a sound file

Key-Value Pairs: There are two key-value pairs:

1. `cName`: The name to associate with the sound object
2. `cDIPath`: The path to the sound file, it may be absolute or relative

```
\appopenDoc({<key-values>});
```

Command Description: Opens a document

[7] The AcroMemory package uses these environments and functions to import icons.

Key-Value Pairs: Here, we list only two of five

1. `cPath`: A device-independent path to the document to be opened. If `oDoc` is specified, the path can be relative to it. The target document must be accessible in the default file system.
2. `oDoc`: (optional) A `Doc` object to use as a base to resolve a relative cPath. Must be accessible in the default file system.

```
\insertPages({<key-values>});
```

Command Description: Inserts pages into the PDF, useful for inserting pages of difference sizes, such as tables or figures, into a LaTeX document which requires that all page be of a fixed size.

Key-Value Pairs: There are five key-value pairs:

1. `nPage`: (optional) The 0-based index of the page after which to insert the source document pages. Use -1 to insert pages before the first page of the document.
2. `cPath`: The device-independent path to the PDF file that will provide the inserted pages. The path may be relative to the location of the current document.
3. `nStart`: (optional) A 0-based index that defines the start of an inclusive range of pages in the source document to insert. If only `nStart` is specified, the range of pages is the single page specified by nStart.
4. `nEnd`: (optional) A 0-based index that defines the end of an inclusive range of pages in the source document to insert. If only `nEnd` is specified, the range of pages is 0 to `nEnd`.

```
\importDataObject({<key-values>});
```

Command Description: Attaches a file to the PDF. This function is used in the two attachments options of AeB Pro.

Key-Value Pairs: There are two key-value pairs of interest:

1. `cName`: The name to associate with the data object.
2. `cDIPath`: (optional) A device-independent path to a data file on the user's hard drive. This path may be absolute or relative to the current document. If not specified, the user is prompted to locate a data file.

```
\executeSave();
```

Command Description: As you know, you must always save your document after it is distilled, this saves document JavaScripts in the document. This command saves the current file so you don't have do it yourself. This command should be the last one listed in the `docassembly` environment.[8]

See the *JavaScript for Acrobat API Reference* [2] for details on these methods and their parameters.

[8] Later commands may dirty the document again, and I have found that saving the document can cause later commands, like `\addWatermarkFromFile`, not to execute.

8.2. Examples

Example 3: Demonstrate \addWatermarkFromFile: The following code places a background graphic on every page the the document. This is the kind of code that is executed for this document.

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

Important: It is *very important* to note that the arguments for this (pseudo-JS method) are enclosed in matching parentheses/braces combination, i.e., ({...}). The arguments are key-value pairs separated by a colon, and the parameters themselves are separated by commas. (The argument is actually an object-literal). It is *extremely important* to have the left parenthesis/brace pair, ({, immediately follow the function name. This is because the environment is a partial-verbatim environment: \ is still the escape, but left and right braces have been "sanitized". The commands, like \addWatermarkFromFile first gobble up the next two tokens, and re-inserts ({ in a different location. (See the aeb_pro.dtx for the definitions.)

Example 4: Demonstrate \getSound: For another cheesy demonstration, let's import a sound, associate it with a button. I leave it to you to press the button at your discretion.

```
\setbox0=\hbox{\includegraphics[height=16bp]{../extras/AeB_Logo.eps}}
\pushButton[\S{S}\W{0}\A{\JS{%
    var s = this.getSound("StarTrek");\r
    s.play();
}}]{cheesySound}{\the\wd0 }{\the\ht0 }

\begin{docassembly}
try {
    \importSound({cName: "StarTrek", cDIPath: "../extras/trek.wav" });
} catch(e) { console.println(e.toString()) };
\end{docassembly}
```

The working version of this appears in aebpro_ex3.

Example 5: Demonstrate \getIcon: Import a few AeB logos (forgive me) and place them as appearance faces for a button. Below is a listing of the code, with some comments added.

```
\begin{docassembly}
// Import the sounds into the document
\importIcon({cName: "logo",cDIPath: "../extras/AeB_Logo.pdf"});
\importIcon({cName: "logopush",cDIPath: "../extras/AeB_Logo_bw15.pdf"});
\importIcon({cName: "logorollover",cDIPath: "../extras/AeB_Logo_bw50.pdf"});
var f = this.getField("cheesySound");   // get the field object of the button
f.buttonPosition = position.iconOnly;   // set it to receive icon appearances
var oIcon = this.getIcon("logo");       // get the "logo" icon
f.buttonSetIcon(oIcon,0);               // assign it as the default appearance
oIcon = this.getIcon("logopush");       // get the "logopush" icon
f.buttonSetIcon(oIcon,1);               // assign it as the down appearance
oIcon = this.getIcon("logorollover");   // get the "logorollover" icon
```

```
f.buttonSetIcon(oIcon,2);              // assign it as the rollover appearance
\end{docassembly}
```

The working version of this appears in aebpro_ex3.

Example 6: Demonstrate \importDataObject: As a final example of docassembly usage, rather than using the attachments options of AeB Pro, you can also attach your own files using the docassembly environment.

```
\begin{docassembly}
try {
    \importDataObject({
        cName: "AeB Pro Example #2",
        cDIPath: "aebpro_ex2.pdf"
    });
} catch(e){}
\end{docassembly}
```

The attachments options automatically assign names. These names appear in the Description column of the attachments tab of Acrobat/Reader. For file attached using the attachsource, the base name plus extension is used, for the files specified by the attachments key, the names are given sequentially, "AeB Attachment 1", "AeB Attachment 2" and so on. When you roll your own, the description can be more aptly chosen. On the other hand, there are commands, introduced later, that allow you to change the default description, to one of your own choosing.

I have found many uses for the execJS environment, or the simplified docassembly environment. You are only limited by your imagination, and knowledge of JavaScript for Acrobat.

9. Linking to Attachments

Should you wish to link to your attachments or *rename their descriptions*, linktoattachments needs to be specified in the option list of aeb_pro. This defines many of the commands discussed in this section enabling you to link to a PDF attachment, open a PDF or non-PDF attachment, save a PDF or non-PDF attachment to the local hard drive, or simply to rename the descriptions of the attachments.

The document aebpro_ex5 has working examples of the ideas and commands discussed in this section.

9.1. Naming Attachments

The description of give an attachment (an embedded file) is used by Acrobat to references its location within the PDF document it is embedded in. This description (or name) is used when creating links to the embedded document as well; consequently, the description is quite important.

• Default Descriptions and Labels

With AeB Pro, you can attach files in three ways: (1) with the attachsource key, (2) with the attachments key; and (3) using the `\importDataObject` method, as demonstrated in Example 6. For attachments that fall into categories (1) and (2), AeB assigns default labels and descriptions. These are presented in Table 1, page 29.

attachsource	label	description
tex	tex	\jobname.tex
dvi	dvi	\jobname.dvi
log	log	\jobname.log
...	...	...
attachments	**label**	**description**
1st file	attach1	AeB Attachment 1
2nd file	attach2	AeB Attachment 2
3rd file	attach3	AeB Attachment 3
4th file	attach4	AeB Attachment 4
...	...	...

Table 1: Default label/descriptions

For documents attached by `attachsource`, the default label is the extension, and the default description is the filename with extension.

For documents attached by the `attachments` option, AeB Pro assigns them "names," which appear in the attachments tab of Acrobat/Reader as the Description.[9] The names assigned are `AeB Attachment 1`, `AeB Attachment 2`, `AeB Attachment 3`, and so on.

If you embed the file using the `docassembly` environment and the `\importDataObject` method (see Example 6, page 28), then you are free to assign a name of your preference.

• Assigning Labels and Descriptions

Whatever method is used to attach a document to the parent document, the names must be converted to unicode on the TEX side of things to set up the links, and there must be a LATEX-like way of referencing this unicode name, hence the development of the `attachmentNames` environment and the two commands `\autolabelNum` and `\labelName`.[10] These two commands, described below, should appear in the `attachmentNames` environment in the preamble.

```
\begin{attachmentNames}
<\autolabelNum and \labelName commands>
\end{attachmentNames}
```

Environment Location: The preamble of the document. The `attachmentNames` environment and the commands `\autolabelNum` and `\labelName` should be used only in the parent document; for child documents they are not necessary.

[9] The Description is important as it is the way embedded files are referenced internally.
[10] It is important to note that these are not needed unless you are linking to the embedded files.

Example 7: Below are the declaration that appear in the supporting file aebpro_ex5:

```
\begin{attachmentNames}
    \autolabelNum{1}
    \autolabelNum*{2}{target2.pdf Attachment File}
    \autolabelNum*[AeST]{3}{AeBST Components}
    \labelName{cooltarget}{My (cool) $|x^3|$ ~ % '<attachment>'}
\end{attachmentNames}
```

Descriptions of these commands follow.

`\autolabelNum[<label>]{<attachment_number>}`

Command Description: For PDFs (or other files) embedded using the `attachments` option, use the `\autolabelNum` command.

Parameter Description: The first optional argument is the label to be used to refer to this embedded file; the default is `attach<attachment_number>`. The second argument is the second is a number, `<attachment_number>`, which corresponds to the order the file is listed in the value of the `attachments` key.[11]

There is a star form of `\autolabelNum`, which allows to to change the description of the attachment.

`\autolabelNum*[<label>]{<attachment_number>}{<description>}`

Command Description: Each file listed as a value in the `attachments` key has a number, `<attachment_number>`, assigned to it according to the order it appears in the list, and a default description of `AeB Attachment <attachment_number>`, This command allows you not only to change the label, but to change the description of the attachment as well.

Parameter Description: The first optional argument is the label to be used to refer to this embedded file; the default is `attach<attachment_number>`. The second argument is the second is a number, `<attachment_number>`. The third parameter is the description that will appear in the attachments pane of Acrobat/Reader.

For files that are embedded in using `\importDataObject`, use the command `\labelName` for assigning the name, and setting up the correspondence between the name and the label.

`\labelName{<label>}{<description>}`

Parameter Description: The first argument is the label to be used to reference this embedded file. The second parameter you can assign an arbitrary name used for the description.

[11] To minimize the number of changes to the document, if you later add an attachment, add it to the end of the list so the earlier declarations are still valid.

• **Notes on the** <description>

The <description> parameter used in \autolabelNum* and \labelName can be an arbitrary string assigned to the description of this embedded file, the characters can be most anything in the Basic Latin unicode set, 0021–007E, with the exception of left and right braces {}, backslash \ and double quotes ". If you take the latin1 option, the unicodes for 00A1–00FF are also included.

A unicode character code can also be entered directly into the description by typing \uXXXX, where XXXX are four hex digits. (Did I say not to use '\'?) This is very useful when using the trouble making characters such as backslash, left and right braces, and double quotes, or using unicode above 00FF (Basic Latin + Latin-1). To illustrate, suppose we wish the description of cooltarget to be

```
"$|e^{\ln(17)}|$"
```

All the bad characters!

```
\labelName{cooltarget}{\u0022$|e^\u007B\u005Cln(17)\u007D|$\u0022}
```

From the unicode character tables we see that

- left brace \u007B
- right brace \u007D
- backslash \u005C
- double quotes \u0022

See the example file aebpro_ex6.tex for additional examples of the use of \uXXXX character codes.

There are several "helper" commands as well: \EURO, \DQUOTE, \BSLASH, \LBRACE and \RBRACE. When the \u is detected, an \expandafter is executed. This allows a command to appear immediately after the \u, things work out well if the command expands to four hex numbers, as it should. Thus, instead of typing \u0022 you can type \u\DQUOTE.

9.2. Linking to Embedded Files

This package defines two commands, \ahyperref and \ahyperlink, to create links between parent and child and child and child. The default behavior of \ahyperref (and \ahyperlink) is to set up a link from parent to child. \ahyperlink and \ahyperref are identical in all respects except for how it interprets the destination. (Refer to 'Jumping to a target' on page 33 for details.)

The commands each take three arguments, the first one of which is optional

```
\ahyperref[<options>]{<target_label>}{<text>}
\ahyperlink[<options>]{<target_label>}{<text>}
```

Command Description: \ahyperref is used to jump to a destination (as specified by the dest key, listed in Table 2, page 32) defined by the \label command, whereas \ahyperlink is used to jump to a destination (as specified by the dest key) defined by the \hypertarget of hyperref. See Section 9.3, page 33 for details.

Parameter Description: The commands each take three arguments, the first one of which is optional.

1. `<options>`: Options for modifying the appearance of the link, and for specifying the relationship between the source and the target file. These are key-value pairs documented in Table 2, page 32.
2. `<target_label>`: The label of the target file, the label is the default label, if there is one, or as defined by `\autolabelNum` or `\labelName`.
3. `<text>`: The text that is highlighted for this link.

Key	Value	**Description**
goto	p2c, c2p, c2c	The type of jump, parent to child, child to parent, and child to child. The default is p2c.
page	<number>	The page of the embedded document to jump to. Default is 0.
view	<value>	The view to be used for the jump. Default is Fit.
dest	<string>	Jump to a named destination. When this key has a value, the values of the keys page and view are ignored.
open	usepref, new, existing	Open the attachment according to the user preferences, a new window, or the existing window. The default is userpref.
border	visible, invisible	Determines whether a visible rectangle appears around the link. The default is invisible.
highlight	none, invert, outline, insert	How the viewer highlights the link when the link is clicked. The default is invert.
bordercolor	r g b	The color of the border when it is visible. The default is black.
linestyle	solid, dashed, underlined	The line style of the border when it is visible. The default is solid.
linewidth	thin, medium, thick	The line width when the border is visible. When invisible, this is set to a width of zero. The default is thin.
preset	<\presetCommand>	A convenience key. You can define some preset values.

Table 2: Key-value pairs for links to embedded files

Example 8: We assume the declarations as given in Example 7, page 30. In the simplest case, we jump from the parent to the first page of a child file given by. . .

```
\ahyperref{attach1}{target1.pdf}
```

This is the same as

```
\ahyperref[goto=p2c]{attach1}{target1.pdf}
```

The goto key is one of the key-value pairs taken by the optional argument. Permissible values for the goto key are p2c (the default), c2p (child to parent) and c2c (child to child).

Example 9: We assume the declarations as given in Example 7, page 30. Similarly, link to the other embedded files in this parent:

```
\ahyperref{attach2}{target2.pdf}
\ahyperref{cooltarget}{aebpro\_ex2.pdf}
```

In all cases above, the \ahyperlink command could have been used, because no *named* destination was specified, without a named destination, the these links jump to page 1.

9.3. Jumping to a target

As you may know, LaTeX, more exactly, hyperref has two methods of jumping to a target in another document, jump to a label (defined by \label) and jump to a target set by \hypertarget. Each of these is demonstrated for embedded files in the next two sections.

The document aebpro_ex5 has working examples of the ideas and commands discussed in this section.

• Jumping to a \hypertarget with \ahyperlink

Suppose there is a destination, with a label of mytarget, defined by the \hypertarget command in target1.pdf. To jump to that destination we would use the following code:

```
\ahyperlink[dest=mytarget]{attach1}{Jump!}
```

Note that dest=mytarget, where "mytarget" is the label assigned by the \hypertarget command in target1.pdf.

• Jumping to a \label with \ahyperref

LaTeX has a cross-referencing system, to jump to a target set by the \label command we use the xr-hyper package that comes with hyperref; the code might be

```
\ahyperref[dest=target1-s:intro]{attach1}
    {Section~\ref*{target1-s:intro}}
```

we set dest=target1-s:intro

The label in target1.pdf is s:intro, in the preamble of this document we have

```
\externaldocument[target1-]{children/target1}
```

which causes xr-hyper to append a prefix to the label (this avoids the possibility of duplication of labels, over multiple embedded files).

9.4. Optional Args of \ahyperref and \ahyperlink

The \ahyperref commands has a large number of optional arguments, see Table 2, page 32, enabling you to create any link that the user interface of Acrobat Pro can create, and more.

These are documented in aeb_pro.dtx and well as the main documentation. Suffice it to have an example or two.

By using the optional parameters, you can create any link the UI can create, for example,

```
\ahyperref[%
    dest=target1-s:intro,
    bordercolor={0 1 0},
    highlight=outline,
    border=visible,
    linestyle=dashed
]{attach1}{Jump!}
```

Now what do you think of that?

The argument list can be quite long, as shown above. If you have some favorite settings, you can save them in a macro, like so,

```
\def\preseti{bordercolor={0 0 1},highlight=outline,open=new,%
    border=visible,linestyle=dashed}
```

Then, we can say more simply, This link is given by...

```
\ahyperref[dest=target1-s:intro,preset=\preseti]{attach1}{Jump!}
```

I've defined a `preset` key so you can predefine some common settings you like to use, the enter these settings through preset key, like so `preset=\preseti`. Cool.

Note that in the second example, `open=new` is included. This causes the embedded file to open in a new window. For Acrobat/Reader operating in MDI, a new child window will open; for SDI (version 8), and if the user preferences allows it, it will open in its Acrobat/Adobe Reader window.

9.5. Opening and Saving with `\ahyperextract`

In addition to embedding and linking a PDF, you can embed most any file and programmatically (or through the UI) open and/or save it to the local file system.

To attach a file to the parent PDF, you can use the `attachsource` or the `attachments` options of AeB Pro, or you can embed your own using the `importDataObject` method, as described earlier in this file.

If an embedded file is a PDF, then you can link to it, using `\ahyperref` or `\ahyperlink`; we can jump to an embedded PDF and jump back. If the embedded file is not a PDF, then jumping to it makes no sense; the best we can do is to open the file (using an application to display the file) and/or save it to the local hard drive.

The AeB Pro package has the command `\ahyperextract` to extract the embedded file, and to save it to the local hard drive, with an option to start the associated application and to display the file. The syntax for `\ahyperextract` is the same as that of the other two commands:

```
\ahyperextract[<options>]{<target_label>}{<text>}
```

Parameter Description: The <options> are the same as \ahyperref (Table 2, page 32), the <target_label> is the one associated with the attachment name, and the <text> is the link text.

In addition to the standard options of \ahyperref, \ahyperextract recognizes one additional key, launch. This key accepts three values: save (the default), view and viewnosave. The following is a partial verbatim listing of the descriptions given in the *JavaScript for Acrobat API Reference*, in the section describing importDataObject method of the Doc object:

1. save: The file will not be launched after it is saved. The user is prompted for a save path.
2. view: The file will be saved and then launched. Launching will prompt the user with a security alert warning if the file is not a PDF file. The user will be prompted for a save path.
3. viewnosave: The file will be saved and then launched. Launching will prompt the user with a security alert warning if the file is not a PDF file. A temporary path is used, and the user will not be prompted for a save path. The temporary file that is created will be deleted by Acrobat upon application shutdown.

Example 10: Here is a series of examples of usage:

1. Launch and view this PDF. The code is
   ```
   \ahyperextract[launch=view]{cooltarget}{aebpro\_ex3.pdf}
   ```
   When you extract (or open) PDF in this way, any links created by \ahyperref or \ahyperlink will not work since the PDF being viewed is no longer an embedded file of the parent.
2. View the a file, but do not save. The code is
   ```
   \ahyperextract[launch=viewnosave]{tex}{aebpro\_ex5.tex}
   ```
   Note that for attachments brought in by the attachsource option, the label for that attachment is the file extension, in this case tex.
3. Save a file without viewing.
   ```
   \ahyperextract[launch=save]{AeST}{AeBST Component List}
   ```

9.6. The child document

If one of the documents to be attached is a PDF document created from a LaTeX source using AeB Pro, and you want link back to either the parent document or another child document, then use the childof option in the aeb_pro option list. The value of this key is the path back to the base name of the parent document. For example, a child document might specify childof={../aebpro_ex5}.

See the support documents aebpro_ex5, the parent document and its two child documents children/target1 and children/target2, found in the examples folder.

10. Creating a PDF Package

The concept of a PDF Package is introduced in Acrobat 8. On first blush, it is nothing more than a fancy user interface to display embedded files; however, it is also used in the new email form

data workflow. Using the new Forms menu, data contained in FDF files can be packaged, and summary data can be displayed in the user interface. Consequently, the way forms uses it, a PDF package can be used as a simple database.

Unfortunately, at this time, the form feature/database feature of PDF Packages is inaccessible to the JavaScript API and AeB Pro. What AeB Pro provides is packaging of the embedded files with the nice UI.

The document `aebpro_ex6` provides a working example of a PDF Package.

To create a PDF Package, embed all files in the parent and use the command `\makePDFPackage` in the preamble to package the attachments.

```
\makePDFPackage{<key-values>}
```

Key-Value Pairs: There are only two sets of key-value pairs

1. `initview=<label>`: Specifying a value for the initview key determines which file will be used as the initial view when the document is opened. If `initview=attach2`, for example, the file corresponding to the label `attach2`, as set up in the `attachmentNames` environment is the initial view. Listing `initview` with no value (or if `initview` is not listed at all) causes the parent document to be initially shown.
2. `viewmode=details|tile|hidden`: The `viewmode` determines which of the three user interfaces is to be used initially. In terms of the UI terminology: `details` = View top; `tile` = View left; and `hidden` = Minimize view. The default is `details`.

If you use this command with an empty argument list, `\makePDFPackage{}`, you create a PDF package with the defaults.

TIP: Use the `\autolabelNum*` command to assign more informative descriptions to the attachments, like so.

```
\autolabelNum*{1}{European Currency \u20AC}
\autolabelNum*{2}{\u0022$|e^\u007B\u005Cln(17)\u007D|$\u0022}
\autolabelNum*[AeST]{3}{The AeBST Components}
\autolabelNum*[atease]{4}{The @EASE Control Panel}
```

Warning: There seems to be a bug when you email a PDF Package. When the initial view is *not a PDF document*, the PDF Package is corrupted when set by email and cannot be opened by the recipient. When emailing a PDF Package, as produced by AeB Pro, always have the initial view as a PDF document.

11. Initializing a Text Field with Unicode

One of the side benefits of the work on linking to attachments of a PDF document is that the techniques are now in place to be able to initialize a text field using unicode characters.

The technique uses a combination of a recently introduced command `\labelName` and a new command `\unicodeStr`.

```
\labelName{<label>}{<string>}
\unicodeStr(<label>)
```

Parameter Description: The parameter `<label>` is a LaTeX-type of label name, and `<string>` is a combination of ASCII characters and unicodes `\uXXXX`, as described earlier (Review the discussion in 'Assigning Labels and Descriptions' on page 29).

Command Description: The command `\unicodeStr` takes its argument, which is *delimited by parentheses*, looks up the string referenced by `<label>` and converts the string to unicode. The unicode tables that come with AeB Pro are used to look up any ASCII characters; for characters that are available on a standard keyboard, unicode escape sequences can be used. This is illustrated below.

For example, we first define a unicode string, and reference it using a label.

```
\labelName{myCoolIV}{\u0022\u20AC|e^\u007B\u005Cln(17)\u007D|$\u0022}
```

Note that the use of `\labelName` *should not occur* within the `attachmentNames` environment, this is for linking to attachments. Here, `\labelName` can be used anywhere before the creation of the text field.

Then we can define a text field with this value as its initial value and its default value like so,

```
\textField[\textSize{10}\textFont{MyriadPro-Regular}
    \uDV{\unicodeStr(myCoolIV)}
    \uV{\unicodeStr(myCoolIV)}
]{myCoolIV}{1.5in}{12bp}
```

The result is the field "€|e^{\ln(17)}|$" Reset

The technique uses special keys as optional arguments of the eforms command `\textField`. The keys `\uDV` and `\uV` signal to the eforms package that the string is given in unicode.

The support document `aebpro_ex8` is a short tutorial on these topics, including additional examples on creating a button and combo box that use unicode encoded strings.

12. Using Layers, Rollovers and Animation.

When the `uselayers` option is taken, the necessary code is input to produce layers (Optional Content Groups). The AcroTeX Presentation Bundle (APB), which has a very sophisticated method of control over layers, by comparison, the AeB Pro layer support is very primitive indeed. As a rule, after you create a layer, you will need a control of that layer. This could be a button or a link that executes JavaScript.

```
\xBld[true|false]{<layer_name>}<content in layer>\eBld
```

Command Description: The basic command for creating a layer is `\xBld`. The content of the layer is set off by the `\xBld`/`\eBld` pair.

Parameter Description: The command `\xBld` takes two parameters: (1) the first is optional, `true` if the layer is initially visible or `false`, the default, if the layer is hidden initially; (2) the name of the layer, this is used to reference the layer, to make it visible or hidden.

A link can be made visible or hidden by getting its OCG object and setting the `state` property. A simple example of this would be. . .

```
\setLinkText[%
\A{\JS{%
    var oLayer = getxBld("mythoughts");
    if ( oLayer != null )
        oLayer.state = !oLayer.state;
    }}
]{\textcolor{red}{Click here}}
```

The link text has a JavaScript action. First we get the OCG object for this layer by calling the `getxBld` function (this is part of the AeB Pro JavaScript) then if non-null (you may not have spelled the name correctly) we toggle the current state, `oLayer.state = !oLayer.state`.

This is such a common action that a formal JavaScript function is defined by AeB Pro

```
\setLinkText[%
\A{\JS{toggleSetThisLayer("mythoughts");}}
]{\textcolor{red}{Click here}}
```

The above examples uses a link, but a form field action can also be used.

An advantage of the layers approach is that the content of the layers are latexed as part of the content of the tex file; consequently, you can include virtually anything in your layer that tex can handle, math, figures, PSTricks, etc. Acrobat Pro 7.0 (and distiller) or later is required to build the layers, but only Adobe Reader 7.0 or later is needed to view the document, once constructed.

12.1. Rollovers

The AeB Pro package offers you two rollovers, which ostensibly provides help to the document consumer.

These topics are illustrated in the support file `aebpro_ex4`.

The `\texHelp` is a command for creating a rollover for some text. When the user rolls over the text, a defined layer is made visible with helpful information. See `aebpro_ex4` for working examples and extensive details.

12.2. Layers and Animation

Let's see if we can conjure up a little animation, shall we?

A working version of this example appears in `aebpro_ex4`.

Example 11: This examples create a sine graph using PSTricks. When the start button is pressed, the function will be graphed in an animated sort of way.

```
\begin{minipage}{.65\linewidth}\centering
\DeclareAnime{sinegraph}{10}{40}
\def\thisframe{\animeBld\psplot[linecolor=red]{0}{\xi}{sin(x)}\eBld}
\psset{llx =-12pt,lly=-12pt,urx =12pt,ury =12pt}
\begin{psgraph*}[arrows=->](0,0)(-.5,-1.5)(6.5,1.5){164pt}{70pt}
    \psset{algebraic=true}%
    \rput(4,1){$y=\sin(x)$}
    \FPdiv{\myDelta}{\psPiTwo}{\nFrames}%
    \def\xi{0}%
    \multido{\i=1+1}{\nFrames}{\FPadd{\xi}{\xi}{\myDelta}\thisframe}
\end{psgraph*}
\end{minipage}\hfill
\begin{minipage}{.3\linewidth}
\backAnimeBtn{24bp}{12bp}\kern1bp\clearAnimeBtn{24bp}{12bp}\kern1bp
\forwardAnimeBtn{24bp}{12bp}
\end{minipage}
```

You will have to delve through the working version of this example in `aebpro_ex4` to fully understand it.

```
\DeclareAnime{<basename>}{<speed>}{<nframes>}
```

This sets the basic parameters of an anime: the base name for the animation, the speed of the animation as measured in milliseconds, and the number of frames to appear in the anime.

```
\animeBld<frame_content>\eBld
```

This `\animeBld`/`\eBld` pair enclose the "i^{th}" frame.

```
\backAnimeBtn[<opts>]{<width>}{<height>}
\clearAnimeBtn[<opts>]{<width>}{<height>}
\forwardAnimeBtn[<opts>]{<width>}{<height>}
```

These are basic button controls for the anime: back, stop/clear, and forward. Each of these has an optional parameter where you can modify the appearance of the button. See the eforms manual for details of these optional parameters.

References

[1] "execJS: A new technique for introducing discardable JavaScript into a PDF from a LaTeX source," TUGBoat, The Communications of the TeX User Group, Vol. 22, No. 4, pp. 265-268 (2001). 24, 25

[2] JavaScript for Acrobat® API Reference, Adobe® Acrobat® SDK, Version 8.0., Adobe Systems, Inc., 2006
http://www.adobe.com/go/acrobat_developer 19, 22, 26

[3] Developing Acrobat® Applications Using JavaScript, Version 8.0., Adobe Systems, Inc., 2006
http://www.adobe.com/go/acrobat_developer

[4] pdfmark Reference Manual, Version 8.0, Adobe® Acrobat® SDK, Version 8.0, 2006
http://www.adobe.com/go/acrobat_developer

[5] PDF Reference, Version 1.7., Adobe Systems, Inc., 2006
http://www.adobe.com/go/acrobat_developer 22

Now, I simply must get back to my retirement. 𝔇𝔖