AcroT_EX.Net

AcroT_EX eDucation Bundle Professional

Enhanced AeB Features using Acrobat Pro

D. P. Story

Table of Contents

1. Forward

For the past several years (this year is 2016), I've been writing a book titled,

AcroTEX eDucation System Tools: LATEX for interactive PDF documents.

The book [6] covers AeB, which includes the eforms package, and AeB Pro in *great detail* and includes many examples to illustrate concepts and techniques. Numerous new examples are available on the CD-ROM that accompanies the book.

During the time of the writing, each of the packages covered was examined, bugs were fixed, and many new and major features were created. Any new features developed in the course of writing the book are documented in the book; however, they are *not included in this documentation.* You can either buy the yet-to-be-submitted book sometime in the future, or discover the features by studying the DTX documentation of the program files. Sorry, it took me three years to write the book, I don't want to spend another year on this documentation. :-{)

Dr. D. P. Story

January 20, 2016

2. Overview

AeB Pro, package file base name aeb_pro (ctan.org/pkg/aeb-pro), is an assortment of features (see Section 2.2 below) implemented through a combination of **pdfmark** operators [4], 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. To have access to all the features of AeB Pro, the document author must have Acrobat Pro 7.0 or later and use dvips/Distiller workflow to create the PDF. For the most part, once the document is assembled, it can be viewed by Adobe Reader 7.0 or later.

non-Distiller workflow

Despite the declaration in the **Forward** to the contrary, this manual will be updated for Version 2.1 of AeB Pro to reflect the creation of a significant new feature, the useacrobat option. For a document author who prefers pdflatex (including lualatex) or xelatex, this option opens the features of AeB Pro provided the document author has the full Acrobat application and has set it up as the primary PDF viewer on his/her computer system. Continue reading about the useacrobat option on page 9. The nopro option has changed as well, the code base that does not depend on the Acrobat application is now available to non-Distiller workflows; refer to the description of the nopro option on page 9 for additional details.

2.1. 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 Adobe 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.

2.2. 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.

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] 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 work-flow, I found them too limiting and too slow in development. For Acrobat, you have a team of top profes-sional 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.

2.3. Sample Files and Articles

The basic distribution demonstration files are available from the AeB Pro website.[5]

AcroTeX Blog. The basic examples from the distribution are also available from the AcroTeX Blog, accessible from the page AeB Pro Demo Files.[6] There is another more recent collection of examples on AcroTeX Blog, these will be referenced in the margin using the icon 🕴, whereas 🕴 refers to a written blog article. In all cases, the source file and any dependent resources are attached to the PDF. A listing of all examples that have the *aeb-pro* tag.[7]

2.4. Requirements

To open this package up to a wider population of users, the requirements for this package have changed; the document author is no longer required to own the Acrobat application (strongly recommended) and is no longer required to use the dvips/Distiller workflow, as previous versions have required. This package classifies you, as the document author, into one of three groups:

1. You own Acrobat and use the dvips/Distiller workflow. This is the ideal workflow for this package because all features of this package are available.

2. You own Acrobat but prefer to use a non-Distiller workflow; that is, you prefer to use the applications pdflatex, lualatex, or xelatex. In this case, almost all features are available through the useacrobat option, refer to the initial description of this option on page 9.

3. You do not own Acrobat, your PDF creator must be pdflatex, lualatex, or xelatex. To avoid compile errors, you must use the nopro option, read the initial description of this option on page 9.

If you do own Acrobat, it must be version 7 or later; to repeat,

> **Acrobat 7.0 Professional** or later is required

If you do not own Acrobat and you want to access the extensive features of AeB Pro beyond what the nopro option provides, you need to buy the application.[8] 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.

AeB Pro requires the insdljs and 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.

---

[5] http://www.math.uakron.edu/~dpstory/aeb_pro.html
[6] http://www.acrotex.net/blog/?cat=98
[7] http://www.acrotex.net/blog/?tag=aeb-pro
[8] In the United States and Europe, Adobe offers a significant academic discount on its software, including Acrobat 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.

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

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, 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).

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!

2.5. The AeB Pro Family of Software

To qualify to be a member of the 'AeB Pro Pro' family, a package must require Acrobat Distiller as the PDF creator. We list many of the members of this exclusive family.

aebxmp is a LaTeX package (ctan.org/pkg/aebxmp) that fills in the advance metadata. The package requires Acrobat 8 Professional and uses E4X, the XML parser that is built into the JavaScript engine.

rmannot (ctan.org/pkg/rmannot) creates rich media annotations (RMA), which may embed or stream SWF, FLV, and MP3 files for playing while a document is being read.

Rich media annotation is a feature of Acrobat/Adobe Reader, version 9 or later. Acrobat Pro and Acrobat Distiller (version 9 or later) are required to build a document, and Adobe Reader (version 9 or later) is needed to activate the annotation and play the media.

AcroFLeX is an application of the rmannot package briefly described above. The acroflex package (ctan.org/pkg/acroflex) creates a graphing screen. The user can type in functions and graph them. A graphing screen can be populated with pre-packaged functions for the user to scrutinize and interact with. The package can graph functions of a single variable x, a pair of parametric equations that are functions of t, and a polar function of t.

The graphing screen is a rich media annotation that uses a specially developed SWF file, called the AcroFₑX Graphing widget. The package takes advantage of rich media annotations, a version 9 feature of Acrobat; it therefore requires Acrobat Pro and Distiller version 9 or later. The user needs to use Adobe Reader (version 9.0 or later) in order to obtain the graphing functionality.

aeb_mlink (ctan.org/pkg/aeb-mlink) creates hypertext links in documents for text extending over *multiple lines*. The package requires that the PDF be created by Acrobat Distiller, version 7.0 or later to create multi-line links, and requires Adobe Reader 7.0 or later for the links to work correctly.

annot_pro (ctan.org/pkg/annot-pro) is used to create text, stamp, and file attachment annotations using Acrobat Distiller that can then be viewed in Adobe Reader.

graphicxsp (ctan.org/pkg/graphicxsp) embeds a graphic file in a PDF document in such a way that the author may reuse that same graphics without significantly increasing the file size. The package also supports the Adobe transparency model.

The next three packages are less important, some are "novelty" packages.

acrosort (ctan.org/pkg/acrosort) is 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.

AeB Slicing is a batch sequence (ctan.org/pkg/aebslicing) 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. It is used for the acromemory package.

acromemory is a LATEX package (ctan.org/pkg/acromemory) 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 CTAN or the AeB Pro family web site:

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

2.6. Package Options

The general syntax for aeb_pro is,

```
\usepackage[⟨options⟩]{aeb_pro}
or
\usepackage[⟨options⟩]{aeb-pro}
```

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

driver=⟨*driver*⟩ The permissible values of ⟨*driver*⟩ are dvips, dvipsone, pdftex, and xetex. The latter two are automatically detected and need not be specified. If no detectable driver is identified and no driver is given, dvips is assumed.

useacrobat For those who prefer to use pdflatex (or lualatex) or xelatex *and* who own the Acrobat application, use the useacrobat option to open all features of this package except for any features associated with the uselayers option. The creation of layers is still only supported through the **pdfmark** operator.

As you go through the examples provided by this package, all sample files work except for the ones using the uselayers or ocganime option.

Continue reading about the useacrobat option in Section 3.

nopro If this option is taken, then no code that requires Distiller or Acrobat is input. Authors who use pdflatex/lualatex/xelatex and who do not own the Acrobat application may have access to the 'nopro' features by taking the nopro option.

Continue reading about the nopro option in Section 4.

AeB Pro Options The AeB Pro package recognizes the components of AeB, these are web, exerquiz, dljslib, eforms, insdljs, eq2db, aebxmp, graphicxsp, 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 5, page 14.

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.

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 10.1, page 28.

attachments This key has its value a list of paths to files to be attached to the parent document. See Section 10.2, page 29.

linktoattachments Invoking this option causes code for linking to attachments, or for giving attachments descriptions other than the default ones. See Section 12, page 42.

childof In a LaTeX child document, use this option to set the path back to the parent document. See Section 12.6, page 49.

btnanime When this option is taken, the code for button animation is included in the compilation. See Section 16, page 54 for details.

uselayers Taking this option brings in code in support of Optional Content Groups, see Section 15, page 51.

ocganime When this option is taken, the code for ocg animation is included in the compilation. See Section 16.2, page 57 for details.

2.7. Installation

AeB Pro requires the installation of AeB (ctan.org/pkg/acrotex). Be sure to install AeB and to read the installation instructions. In this section, we outline the method of installing AeB Pro.

• **Automatic installation**

Some TeX systems, most notably **MiKTeX** and **TeX Live**, have a Package Manager to automatically download and install LaTeX packages. If you have a Package Manager and not installed AeB Pro do so now. After AeB Pro is installed, it is not quite ready to be used. Locate where the Package Manager installed the documentation portion of the installation, for MiKTeX, this might be at

```
C:\Program Files (x86)\MiKTeX 2.9\doc\latex\aeb-pro
```

(This path assumes the use of **MiKTeX 2.9**.) The folder contains documentation and example files.

• **Manual installation**

Manual installation may be necessary for some TeX systems, or for the case where you have downloaded the ZIP package files from CTAN or from the home website of AeB Pro at http://www.math.uakron.edu/~dpstory/aeb_pro.html.

The AeB Pro distribution comes in two ZIP files: `aebpro_pack.zip` and `aebpro.zip`. The first contains the program files and documentation,[9] the latter contains the full distribution, including program files, documentation, and example files. If you already have AeB Pro, it suffices to update your installation using `aebpro_pack.zip`. If you don't have AeB Pro already installed, the install the contents of `aebpro.zip`.

To install AeB Pro, use the following steps:

1. Place `aebpro.zip` (or possibly `aebpro_pack.zip`) on your latex search file and unzip. (If you already have an `aeb_pro` folder, unzip one level above the `aeb_pro` folder.) Unzipping creates a folder named `aeb_pro`.

   Installing AeB Pro with MiKTeX 2.8 or later. MiKTeX 2.8 or later is more particular about where you install packages by hand. If you are installing AeB Pro by hand, MiKTeX 2.8/2.9 requires that you install the distribution in a local root TDS tree. Review the MiKTeX help page on this topic

   http://docs.miktex.org/manual/localadditions.html

   Within `C:\Local TeX Files\tex\latex`, copy `aebpro_pack.zip` (and possibly `aebpro.zip`) and unzip. Unzipping creates a folder named `aeb_pro`.

   If you already have AeB Pro that was automatically installed on your MiKTeX system, you should delete this old version of AeB Pro. You may have to use the MiKTeX package manager to remove them from the MiKTeX database registry.

---

[9]Available from http://www.math.uakron.edu/~dpstory/aeb_pro.html

2. Within the `aeb_pro` folder, latex the file `aeb_pro.ins` file, this unpacks the installation.

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

3. Install the JavaScript file, `aeb_pro.js`, as explained in the next subsection.

AeB (AcroTEX eDucation Bundle) is also required, installation instructions are contained in the AeB reference document, the instructions are reproduced here for your convenience.

To install AeB, use the following steps:

1. Place `acrotex.zip` in your latex search file and unzip. (If you already have an `acrotex` folder, you should unzip the file `acrotex.zip` one level above the `acrotex` folder.) Unzipping creates a folder named `acrotex`.

   Installing AeB with MiKTEX 2.8 or later. MiKTEX is more particular about where you install packages by hand. If you are installing AeB by hand (recommended), MiKTEX requires that you install the distribution in a local root TDS tree. Review the MiKTEX help page on this topic

   http://docs.miktex.org/manual/localadditions.html

   Within the local root folder, e.g., `C:\Local TeX Files\tex\latex`, copy the file `acrotex.zip` and unzip it. Unzipping creates a folder named `acrotex`.

   If you already have AeB that was automatically installed on your MiKTEX system, you should delete this old version of AeB. You may have to use the MiKTEX Package M anager to remove them from the MiKTEX database registry.

2. Within the `acrotex` folder, latex the file `acrotex.ins` file, this unpacks the installation.

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

3. Install the JavaScript file, `aeb.js`, as explained in the next subsection.

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

In previous versions of this package, the JavaScript support files `aeb_pro.js` and `aeb.js` where distributed with `aeb_pro`. Now they are distributed through the package acrotex-js;[10] these JavaScript support files are found in the `js-files` folder of the *acrotex-js pkg* acrotex-js distribution. The acrotex-js package, dated 2021/06/19 or later, is required by `aeb_pro`; you will have to install it, or, some TEX systems automatically download and install the package for you, since it is required.

The instructions for installing the JavaScript support files `aeb_pro.js` and `aeb.js` are in the document `install_jsfiles.pdf`, which resides in the `docs` folder of the acrotex-js installation.

---

[10]https://www.ctan.org/pkg/acrotex-js

After you've installed the JavaScript files, as directed by the `install_jsfiles.pdf` document, validate the installation of the JavaScript files by navigating to the `js-test` folder of the acrotex-js package and opening the file `test_install.pdf` in Acrobat; follow the directions contained therein.

Configure Acrobat DC

Adobe Acrobat DC authors. If you have Acrobat DC that was purchased or updated after December 2020, Acrobat needs to be configured to function correctly. For detailed discussion, refer to the document `acrobat-in-workflow.pdf` found in the doc folder of this distribution.

• Installing `aebpro.cfg`

The distribution comes with a file named `aebpro.cfg`, the contents of which are displayed below:

```
%
% AeB Pro Configuration file
%
\ExecuteOptionsX{driver=dvips}
```

Locate this file in the root folder of the AeB Pro installation. If the driver is specified in the configuration file, it need not be included in the option list of `aeb_pro`.

2.8. 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 full screen 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 various attachment 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.

These sample files are available in the `examples` folder of the `aeb_pro` folder. When links given above are to the same files on the AcroTeX Blog.

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. Alternatively, if the above links do not work because you have moved this documentation, you can also access the basic distribution examples from the AcroTeX Blog website, refer to the named paragraph **AcroTeX Blog** on page 6 for links to this resource.

3. Concerning the `useacrobat` option

For document authors who have the full Acrobat application but prefer a non-Distiller workflow, use the `useacrobat` option to declare to AeB Pro that you have Acrobat. Traditionally, if neither pdflatex (including lualatex) nor xelatex are used, a dvips/ Distiller workflow is assumed. Declaring the `useacrobat` option opens up all features except for those that depend on the use of layers (also called OCGs); in particular, the options `uselayers` and `ocganime` are disallowed. All features, therefore, described in this manual are available to pdflatex, lualatex, and xelatex document authors who have Acrobat, *except for those features* described in Section 15 and Section 16.2.

Syntax. The syntax for this option is simple:

```
\usepackage[useacrobat,
  ⟨options⟩
]{aeb_pro}
```

where ⟨*options*⟩ *does not include* the `uselayers` and `ocganime` options.

Regarding AeB Control Central (Section 5), when you specify `useacrobat`,[11] the packages graphicxsp and rmannot of display (1) are not allowed to be specified in the option list of aeb_pro. These two packages use the **pdfmark** operator to implement their features; a dvips/Distiller workflow is required in this case.

4. The nopro option

For authors *who do not have* the Acrobat application, you must use pdflatex (or lualatex) or xelatex to produce your PDF. The only option for using AeB Pro is to specify the nopro option.

Syntax. The syntax for nopro is to declare it in the option list:

```
\usepackage[nopro,
  ⟨options⟩
]{aeb_pro}
```

---

[11] This implies you are not using `Distiller` as the PDF creator.

Specifying this option excludes the use of layers (the options `uselayers` and `ocganime` are prohibited) and any commands or environments that use 'post-PDF creation' methods will silently (or not so silently) fail. All features are available that are implemented using LaTeX that does not depend on the driver, or LaTeX that is driver dependent. These include the commands and environments of Sections 5–9.

5. 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 AeB components supported by AeB Pro are listed below.

$$\begin{array}{llllll} \text{web} & \text{exerquiz} & \text{dljslib} & \text{eforms} & \text{insdljs} \\ \text{eq2db} & \text{aebxmp} & \text{hyperref} & \text{graphicxsp} & \text{rmannot} & \text{aeb\_mlink} \end{array} \tag{1}$$

The phrase 'AeB Control Central' refers to using these package names as options for the `aeb_pro` package. 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=dvips,
    web={pro,designv,tight,usesf,req=2016/11/03},
    exerquiz={⟨pkg-opts⟩},
    ...,
    aebxmp
]{aeb_pro}[2016/12/10]
```

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

Beginning with `aeb_pro` dated 2016/12/10, a new `req` key is defined for the values of the AeB packages listed in display (1). Using the `req` you can specify the release date (required date) of the AeB package being used; this assures that the document will compile and any new JavaScript code is present. In the example above, the `web` package is required for release date of 2016/11/03 (or later); the `aeb_pro` release date of 2016/12/10 is required, for that is the date this `req` key was implemented. The `req` key is optional.

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

xetex, for example. Of course, if you own the Acrobat application, a subset of features are available, as discussed Section 3, using the useacrobat option.

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

人📄 The support file aebpro_ex2 has a section discussing the AeB Control Central, as well as features of the extended (pro) option of the web package.

6. Declaring the Initial View

The \DeclareInitView command is a "data structure" for setting the Initial View of the Document Properties dialog box, see Figure 1. \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 ⟨pos num⟩	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	⟨pos num⟩	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 (Figure 1). 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 (Figure 1) of the Document Properties dialog box.
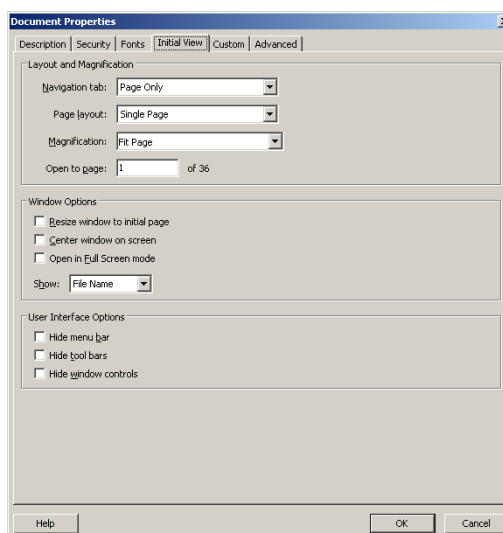
Figure 1: Initial View of Document Properties

\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. The \DeclareInitView commands populates the Initial View tab of the Document Properties dialog box, shown in Figure 1.

7. Document Actions

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

7.1. Document Level JavaScripts

The creation of document level JavaScript has been a 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.

7.2. Set Document Actions

The AeB Pro provides environments for the Acrobat events willClose, willSave, did-Save, 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}
   ⟨script⟩
\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}
   ⟨script⟩
\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}
   ⟨script⟩
\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}
   ⟨script⟩
\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}
   ⟨script⟩
\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.

Developer Notes: I've inserted five commands,

```
\developer@will@Close
\developer@will@Save
\developer@did@Save
\developer@will@Print
\developer@did@Print
```

that are let to \@empty. A package developer can insert JS code to make the package behave as desired, while the document author can use the above environments to add any additional scripts.

The correct way of using these commands is

```
\begin{defineJS}{\my@WillClose}
    ⟨some willClose script⟩
\end{defineJS}
\let\my@save@developer@will@Close\developer@will@Close
\def\developer@will@Close{%
    \my@save@developer@will@Close
    \my@WillClose
}
```

This is the technique I used in the acroflex package.

7.3. Document Open Actions

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

```
\additionalOpenAction{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,

```
\additionalOpenAction{\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 \requires-Version command.

```
\requiredVersionMsg{message}
\alternateDocumentURL{url}
\requiredVersionMsgRedirect{message}
\afterRequirementPassedJS{JS code}
\requiresVersion[option]{version_number}
```

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. If the keyword warnonly is passed as the value of the optional parameter, the alert messages will appear, but the file will not be closed or redirected.

Important: The command \requiresVersion needs to be issued *after* any redefinitions of the

> \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 \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 \additionalOpenAction; 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.

8. 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.

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

8.1. 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{script}}
```

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.

```
\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 ⟨script⟩ is executed each time the page is opened or closed.

```
\begin{addJSToPageOpen}
    ⟨script⟩
\end{addJSToPageOpen}
```

For page close events, we have the addJSToPageClose environment.

```
\begin{addJSToPageClose}
    ⟨script⟩
\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}
```

8.2. 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}
    ⟨script⟩
\end{addJSToPageOpen}
```

```
\begin{addJSToPageClose}
    ⟨script⟩
\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}
    ⟨script⟩
\end{addJSToPageOpenAt}
```

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

```
\begin{addJSToPageCloseAt}{page-ranges}
    ⟨script⟩
\end{addJSToPageCloseAt}
```

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

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 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.printlnstr;
\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.printlnstr;
\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.

8.3. 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}
   ⟨script⟩
\end{everyPageOpen}
```

For page close events, we have the everyPageClose environment.

```
\begin{everyPageClose}
   ⟨script⟩
\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: These two commands cancel the current everyPageOpen and everyPageClose events. Following the cancel commands, use the everyPageOpen or everyPageClose environment to create different every page events.

9. Fullscreen Support

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

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

9.1. Set 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{`*KV-pairs*`}`

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 `\setPageTransition`).

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.

9.2. 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{KV-pairs}
```

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: 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}
```

\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}{KV-pairs}
```

Key-Value Pairs: Same as \setPageTransitionAt

Parameter Description: The parameter *page-ranges* has the same format as described in Section 8.2, page 22. This command obeys the \atPage.

For example,

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

10. 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 a file with the same base name as \jobname, that is a file of the form \jobname.*ext*.

2. attachments is a general option for attaching a file, as specified by its absolute or relative path.

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

10.1. 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).

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!

10.2. 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!

10.3. Optional attachments

The `attachments` options allows you to list a collection of files to be attached to the PDF file. The optional attachments concept allows you to develop a list of attachments that are attached if the `optattachments` option is taken and not attached if the `!optattachments` option is taken. AeB Pro defines \ifoptattachments, a Boolean switch, which is set to true by `optattachments` and to false by `!optattachments`.

```
\addtoOptAttachments{list-of-files}
```

This command can be used anywhere in the document and adds the listed files to the ones to be attached. The list is comma delimited.

The initial application and motivation of this feature is for projects that are worked on by a team of LaTeX authors. The master project file inputs sections of the document using the \input command of LaTeX. The request was for any file that is input using \input be added to the optional attachments list; therefore, the AeB Pro package defines the following two commands:

```
\prjinput{file}
\prjinclude{file}
```

These are the "project" version of the LaTeX commands \input and \include, respectively. Each of these adds *file* to the list of optional attachments, then passes the argument to the two user commands \prjInputUser and \prjIncludeUser, respectively.

```
\newcommand{\prjInputUser}[1]{\input{#1}}
\newcommand{\prjIncludeUser}[1]{\include{#1}}
```

Above you see the default definitions of these user commands. They may be redefined as desired to achieve some special effect.

```
\renewcommand{\prjInputUser}[1]{%
  \marginpar{\fbox{\footnotesize\ttfamily#1}}
  \input{#1}}
```

It is possible to define \prjInputUser to be a link or button that opens the attached file.

When the !optattachments is taken, the default definitions of the two commands \prjinput and \prjinclude are \prjInputUser and \prjIncludeUser, respectively; this bypasses the step of adding the file to the list of optional attachments. The switch \ifoptattachments can be used, as needed, in a custom definition of \prjInputUser or \prjIncludeUser; for example,

```
\renewcommand{\prjInputUser}[1]{%
  \marginpar{\fbox{\footnotesize\ttfamily\ifoptattachments
  \setLink[\A{\JS{this.exportDataObject({%
    cName: "\getcNameFromFileName{#1}",nLaunch: 0});}}
  ]{\textcolor{\ahrefcolor}{#1}}\else#1\fi}}\input{#1}}
```

This code creates a link that saves the attachment, if optattachments is taken, and simply puts the file name the margin, if !optattachments is taken. The JavaScript method this.exportDataObject is used to extract and save the attachment; the command \getcNameFromFileName is an AeB Pro command that associates the file name (#1) with the attachment label name (the cName key).

11. 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}
  ⟨script⟩
\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 ⟨*script*⟩ is *executed one time and then discarded*, see [1] for an article on this topic. This technique only works if you have Acrobat Pro.

11.1. 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({KV-pairs});
```

Command Description: Inserts a watermark into the PDF

Key-Value Pairs: Numerous, see the `addWatermarkFromFile()` method [2]. 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({KV-pairs});
```

Command Description: Imports icon files[12]

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({KV-pairs});
```

Command Description: Imports a sound file

---

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

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({KV-pairs});
```

Command Description: Opens a document

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({KV-pairs});
```

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({KV-pairs});
```

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.[13]

```
\sigInfo{
    cSigFieldName: "mySig",
    ohandler: security.PPKLiteHandler,
    cert: "D_P_Story.pfx",
    password: "dps017",
    oInfo: {
        location: "Niceville, FL",
        reason: "I am approving this document",
        contactInfo: "dpstory@acrotex.net",
        appearance: "My Signature"
    }
};
\signatureSign
```

Command Description: The eforms package supports the creation of signature fields. Such fields can be signed using the Acrobat UI, or programmatically using the `\sigInfo` and `\signatureSign` commands. See the eforms manual, `eformman.pdf` for a detailed description of the parameters of `\sigInfo`.

```
\signatureSetSeedValue(oSeedValue)
```

Command Description: The Acrobat JavaScript methods *Field*.signatureSetSeed-Value is implemented through the LaTeX comment `\signatureSetSeedValue`. The method needs the field object of the signature field, this is passed to `\signatureSet-SeedValue` through the JavaScript variable `oSigFileName`. To use this command, you first get `oSigFileName`, like so,

---

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

```
var sv={
    mdp: "defaultAndComments"
    reasons: ["This is a reason","This is a better reason"],
    flags: 8
};
var oSigFileName=this.getField("sigOfDPS");
\signatureSetSeedValue(sv);
```

The above code defines a object, sv, with seed value properties: the implication of the mp entry, is that the signature field is now a certification signature, filling in form fields and making comments do not invalidate the signature; when the user signs the document, he must choose from the two listed reasons, and none other; the flags property makes the choice of a reason a requirement. The next line, following the definition of sv, we get the field object of the signature field, and name it oSigFileName, this is the name that \signatureSetSeedValue uses. Finally, we pass the sv object to \signatureSetSeedValue.

Additional information on signatures can be found at the Acrobat Developer Center.[14] The *JavaScript for Acrobat API Reference* [2] for details on these methods and their parameters.

11.2. Examples

Example 3: Demonstrate \addWatermarkFromFile: The following code places a background graphic on every page the document. This is the kind of code that is executed for this document. The value of the cDIPath parameter requires an absolute path to the watermark file.

```
% Add a watermark from a folder different from the source file
\begin{docassembly}
\addWatermarkFromFile({
  bOnTop:false,
  cDIPath: "/C/AcroPackages/ManualBGs/Manual_BG_DesignV_AeB.pdf"
});
\end{docassembly}
% Add a watermark from the source folder
\begin{docassembly}
\retnAbsPathAs(_path); % Helper command that acquires the abs path
_path += "myWaterMark.pdf";
\addWatermarkFromFile({
  bOnTop:false,
  cDIPath: _path
});
\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

---

[14]http://www.adobe.com/go/acrobat_developer

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 De-
scription 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.

11.3. Pre-`docassembly` Methods

In this section, we'll gather some "useful" commands that may be useful in combin-
ing several `docassembly` tasks together. The `docassembly` environment is a partial-
verbatim environment, expansion is severely limited. The trick is to expand before
placing the lines in the `docassembly` environment.

• Importing and Placing Images

In this section we introduce four commands for importing images (possibly with various
graphic formats) into the PDF document, and inserting them as images that appear in
the document itself. These are

> `\declareImageAndPlacement, \declareMultiImages,`
> `\insertPreDocAssembly,` and `\placeImage.`

The file `placeimages.pdf` is a demo of the commands of this section. The source file
and images are attached to the PDF. The PDF, titled *Importing and Placing Images using
AeB Pro*, is found at the AcroTeX Blog web site.

`\declareImageAndPlacement{`*KV-pairs*`}`

Command Description: This command creates the JavaScript code to import images
(icons) and associates them with target push buttons that are created by `\placeImage`.
The images are imported into the document when the document is first opened in
Acrobat. The images themselves can be PDF, BMP, GIF, JPEG, PCX, PNG, TIFF, or some
other supported format. (See the documentation on the method *Doc*.`importIcon` for
details.) The file is converted to PDF when imported.

Important: This command is executed in the preamble only, can be executed more than
once, (once for each image being imported), and *outside of the `docassembly` environ-
ment.*

Key-Value Pairs: This command takes up to four key-value pairs.

- `name`: (*Optional*) The symbolic name to be associated with this image. The name is later used to attach the image to the push button (created by `\placeImage`). If a value for this key is not provided, one will be automatically created.

- `path`: (*Required*) The path to the image, this can be a relative path or an absolute path. If absolute, use the device independent path notation of Acrobat; for example,

      /C/acrotex/myimages/myimage.png.

- `page`: (*Optional*) If the image is a PDF, the PDF may contain several pages, each with an image on it. You can specify which of the pages to import using the `page` key. If no `page` key is specified, page 0 is assumed.

- `placement`: (*Optional*) This is a comma-delimited list of names of push buttons created by the command `\placeImage`. Multiple place images with the same name all get the image imported into it. If you want several place images with different names, list these in a comma delimited list, like so

      placement={image1,image2,image3}.

  If a value for this key is not provided, a message in the log is generated; the images are imported (embedded) in the document, are not used to create visible images. Either provide a `placement` key-value pair, or learn how to use (named) icons that are embedded in the document with the *Doc*.`importIcon()` method.

Multiple images can be imported and set by simply executing `\declareImageAndPlacement` multiple times with a different set of arguments, or, by executing `\declareMultiImages`.

```
\declareMultiImages
{
   {KV-pairs_1}{KV-pairs_2}{KV-pairs_3}...{KV-pairs_n}
}
```

Command Description: This command calls the `\declareImageAndPlacement` command for each of its arguments. The argument of the command is a series of tokens (enclosed in braces); within each pair of matching braces are *KV-pairs* of the arguments of `\declareImageAndPlacement`. The command `\declareMultiImages` loops through the list, calling `\declareImageAndPlacement` with the current set of *KV-pairs*. An example follows.

```
\declareMultiImages
{%
   {path=graphics/girl.png,placement={Avatar3,Avatar4}}
   {path=graphics/AcroFord.jpg,placement=AcroFord}
   {path=graphics/scot.gif,placement=Scot}
}
```

Important: This command is executed in the preamble only, can be executed more than once, (once for each image being imported), and *outside of the docassembly environment.*

Once the images have been defined and referenced using any of the above commands, you need to actually executed the JavaScript code these commands created. This is done with the \insertPreDocAssembly inside the docassembly environment.

```
\insertPreDocAssembly
```

Command Description: This command expands to all the JavaScript code created by the commands \declareImageAndPlacement and \declareMultiImages. It is *placed within the docassembly environment*, like so

```
\begin{docassembly}
\insertPreDocAssembly
\end{docassembly}
```

The target buttons are be created by the \pushButton command from the eforms package, but as a convenience, AeB Pro defines the \placeImage command

```
\placeImage[options]{name}{width}{height}
```

Command Description: Creates a readonly push button with an icon only appearance. The *options* can be used to modify the appearance of the button (add a border, for example); the *name* is used as the field name, and is referenced in the placement key of \declareImageAndPlacement. The *width* and *height* are what they appear to be, the width and height of the image.

If the width and height is not correct, Acrobat will scale the image. There are other controls that can be used through the optional arguments to position the image within its bounding box. The dimensions of the image you want to use can be acquired through various methods. On windows, the dimension of the image for PNG and JPG are displace when the mouse is moved over the image (while using explorer).

Note. At times, I have imported images this way, these commands just simply the task. This method may be preferred over using \includegraphics when the image has transparency that you want to preserve. For example, if a PNG image has a transparent background, it will be imported into the document with the transparent background. For those using Adobe Distiller, the transparency is often lost (unless the image uses vector graphics) when converting to an EPS file.

- **Creating Custom Button Appearances**

The \placeImage command described in the last section is a \pushButton designed to be read only and is meant to be used to place non-interactive images in the document. The methods of the previous section can also be used to create custom button appearances using graphics files of various formats.

To this end, an optional parameter is defined for the value(s) of the placement parameter in \declareImageAndPlacement and \declareMultiImages. Each button

has (at most) three appearance states: normal, rollover, and down. The additional optional parameter allows you to specify what appearance state the icon is to be used for. The optional parameter is shown in the example below.

```
\declareMultiImages
{%
   {path=graphics/man1.pdf,placement={Avatar1,[2]Avatar2}}
   {path=graphics/scot.gif,placement={[1]Avatar1,[1]Avatar2}}
   {path=graphics/girl.png,placement={[2]Avatar1,[0]Avatar2}}
}
```

The optional argument precedes the field name and determines the appearance state of the button the icon is to be used; the values are [0] (default, normal icon); [1] (down icon); and [2] (rollover icon). There must be no space between the optional argument and the field name; if you type "[2] Avatar1", for example, the field name is interpreted as " Avatar1", which is incorrect.

⊛ Further details and examples may be found in the blog article button_appr.pdf titled *Creating Button Appearances*, found at the AcroTeX Blog web site. The source file and images are attached to the PDF.

• Methods in support of Button Anime

The commands and methods described in this section are in support for 'Button and Ocg Anime' on page 54.

The btnanime option brings in the code necessary to create what I call button anime, as opposed to OCG anime (using layers). Two pre-docassembly commands were created for this purpose \embedMultiPageImages and \placeAnimeCtrlBtnFaces.

```
\embedMultiPageImages{options}
```

Command Description: The command embeds (in the document) and optionally places a series of icons all of which are pages of the same PDF file. This command does not need the btnanime option, it is part of the core AeB code.

Key-Value Pairs: There are a number of key-value pairs the argument recognizes.

- path: The path to the PDF containing the images to be used. This path can be relative or absolute. This key is required.

- name: The base name to be associated with the images being embedded. This key is required.

- placement: A comma-delimited list, each member of the list is the base name of the button anime field created by \btnAnime that is to use this set of images. This key is optional, if not present, the images are embedded only.

- firstpage: The page number (the first page is page 1) of the first image to be embedded and used. This key is optional; if firstpage is not specified, its default value is 1.

- `lastpage`: The page number (base-1 page numbering) of the last image to be embedded and used. The value of this key is required.

For example, here we embed the first 41 pages from the file `sine_anime.pdf`, which resides in the subfolder `graphics`, and associates it with the button anime field `mysine`.

```
\embedMultiPageImages{lastpage=41,name=sine,
    path=graphics/sine_anime.pdf,placement=mysine}
```

The control buttons for a button anime require a custom appearance. AeB Pro supplies one set of custom icons, and allows for the creation of more by the interested document author, that's you.

```
\placeAnimeCtrlBtnFaces[path]{⟨appr_icons⟩.pdf}{list_of_animes}
```

Command Description: This command associate the icon set to be used for the button appearances of the buttons used to control the button animation. (That's a lot of buttons in that last sentence!)

Parameter Description: The command takes three parameters, one of which is optional.

[*path*] is the (optional) path to the button control appearance icons. If no argument is present, the value of \pathToBtnCtrlIcons is used, see `aebpro.cfg` for its definition. The icon set distributed with AeB Pro is named `btn_anime_icons1.pdf`, and is found in the icons folder of the `aeb_pro` folder.

⟨*appr_icons*⟩ is the name of the PDF file that contains the icons to be used for the appearances of the control buttons.

list_of_animes is a comma-delimited list of the base names of the anima created by \btnAnima that will be using these appearance icons.

Example:

```
\placeAnimeCtrlBtnFaces{btn_anime_icons1.pdf}{myclock,mysine}
```

The above example uses no optional parameter, so \placeAnimeCtrlBtnFaces uses the path defined by \pathToBtnCtrlIcons in `aebpro.cfg`. On my personal system, the `aebpro.cfg` file reads

```
%
% AeB Pro Configuration file
%
\ExecuteOptionsX{driver=dvips}
\renewcommand{\pathToBtnCtrlIcons}
    {C:/Users/Public/Documents/My TeX Files/tex/latex/aeb_pro/icons}
```

You should seek out the `aebpro.cfg` and edit this command to point to the `icons` folder of `aeb_pro`. You can create your own icon PDF file in the `icons` folder; the guidelines for creating such an icon PDF file are simple. Places the icon from each page

as the appearance of a corresponding control button. The expected order of the icons is given in Figure 2, page 41.

   If you put your custom appearance icon set in the source folder, you can reference like so,

```
\placeAnimeCtrlBtnFaces[.]{btn_anime_custom.pdf}{myclock,mysine}
```

The optional [.] overrides the definition of \pathToBtnCtrlIcons, and refers to the current folder. If you never defined \pathToBtnCtrlIcons (its default definition is empty), then

```
\placeAnimeCtrlBtnFaces{btn_anime_custom.pdf}{myclock,mysine}
```

does the trick. If you have them in a subfolder (graphics) of the source file folder, then

```
\placeAnimeCtrlBtnFaces[graphics]{btn_anime_custom.pdf}{myclock}
```

Page	Icon
0	Go to first frame
1	Go to last frame
2	Step back one frame
3	Step forward one frame
4	Play backward
5	Play forward
6	Pause
7	Increase speed
8	Decrease speed

Figure 2: Icons by page for Button Anime Controls

   Finally, because these are pre-docassembly methods, these two commands go in the preamble, and are followed by the docassembly environment; like so,…

```
\embedMultiPageImages{lastpage=36,name=rotate,
    path=graphics/animation.pdf,placement=myAnimation}
\embedMultiPageImages{lastpage=41,name=sine,
    path=graphics/sine_anime.pdf,placement=mysine}
\placeAnimeCtrlBtnFaces{btn_anime_icons1.pdf}{myAnimation,mysine}
\begin{docassembly}
\insertPreDocAssembly;
\executeSave;
\end{docassembly}
```

   You may have other pre-docassembly commands as well.

12. Linking to Attachments

Should you wish to link to your attachments or *rename their descriptions*, the linkto-attachments option 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.

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

12.1. Naming Attachments

Associated with each embedded file is a name (or label) and a description. The name given to an attachment (an embedded file) is used by Acrobat to reference its location within the embedding PDF document. This name (or label) is used when creating links to the embedded document as well; consequently, the assigned name 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 42.

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.[15] The names assigned are AeB Attachment 1, AeB Attachment 2, AeB Attachment 3, and so on.

If you embed a file using the \importDataObject method within the docassembly environment (see Example 6, page 35), you are free to assign a name of your preference.

---

[15]The Description is important as it is the way embedded files are referenced internally.

- **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 `attach-mentsNames` environment and the two commands `\autolabelNum` and `\labelName`.[16] 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.

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}{{\AEB}ST Components}
    \labelName{cooltarget}{My (cool) $|x^3|$ ~ % '<attachment>'}
\end{attachmentNames}
```

Descriptions of these commands follow.

```
\autolabelNum[label]{⟨num⟩}
```

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 ⟨*num*⟩. The second argument is the second is a number, ⟨*num*⟩, which corresponds to the order the file is listed in the value of the `attachments` key.[17]

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

```
\autolabelNum*[label]{⟨num⟩}{⟨description⟩}
```

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

---

[16]It is important to note that these are not needed unless you are linking to the embedded files.

[17]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.

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

For files that are embedded 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, ⟨*description*⟩, you can assign an arbitrary name used for the description.

• **Notes on the ⟨*description*⟩ parameter**

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 ".

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

人📄 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.

12.2. 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 46 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 46) 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 12.3, page 46 for details.

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

> *options* are the 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 46.
>
> *target_label* is the label of the target file, the label is the default label, if there is one, or as defined by \autolabelNum or \labelName.
>
> *text* is the text that is highlighted for this link.

Example 8: We assume the declarations as given in Example 7, page 43. 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 43. 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.

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 in-visible.
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

12.3. 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.

人📄 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).

12.4. Optional Args of `\ahyperref` and `\ahyperlink`

The `\ahyperref` commands has a large number of optional arguments, see Table 2, page 46, 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.

12.5. 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, 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 those for `\ahyperref` (refer to Table 2, page 46), 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}{{\AEB}ST Component List}
   ```

12.6. 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}`.

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

13. 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.

人📄 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 `\make-PDFPackage` in the preamble to package the attachments.

```
\makePDFPackage{KV-pairs}
```

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

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, for example, initview=attach2, 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.

viewmode=⟨details|tile|hidden⟩ The viewmode determines which of three user interfaces is to be used initially. In terms of the UI terminology:

> details = View top
>
> tile = View left
>
> 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 {\AEB}ST 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.

14. 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}
\defUniStr{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 43).

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}
  \DV{\unicodeStr(myCoolIV)}\V{\unicodeStr(myCoolIV)}
]{myCoolIV}{1.5in}{12bp}
```

The result is the field

For version 2.9 of aeb_pro, the package forms16be is included when the option linktoattachments is specified. The package gives support to initializing form fields, as just described. In that package, \defUniStr is defined and may be used instead of \labelName. See the documentation and sample files of forms16be for more information on the topic of initializing fields using 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.

15. 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|print=⟨true|false⟩]{layer_name}
  ⟨content of 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. The print key sets the printing attribute of the of the layer:

- `print=true` (or just `print`): the layer *always prints*, no matter if it is visible or not.

- `print=false`: the layer *never prints*, no matter if it is visible or not.

- If the print key does appear in the list of optional parameters, the layer will print if visible, otherwise, it does not print. Normally, the `print` key is not specified, and the layer is printed only if visible.

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.

🌐 The file `xbld_options.pdf` is a demo of the optional parameters for `\xBld` of this section. The source file and images are attached to the PDF. The PDF, titled *Exploring the options of \xBld*, is found at the AcroTeX Blog web site.

15.1. Rollovers

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

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

🌐 Additional examples of *rollover animation* using `\texHelp` can be found at the AcroTeX Blog web site.

15.2. Using the form field tool tip feature

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.

15.3. Layers and Animation

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

人圖 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.

16. Button and Ocg Anime

In this section. we introduce some commands and one environment for creating button and OCG anime (see page 57).

16.1. The `\btnAnime` Command

When animating with layers, we create a series of frames in different layers, and we animate by successively making visible then hiding each of the layers in turn. The same can be done with buttons. Buttons can take on appearances using what I'll call icons. We create a series of stacked buttons, each with an icon appearance; when the animation is started, each button becomes visible and hidden in turn.

The demo file for the material in this section, (and for the support material in 'Methods in support of Button Anime' on page 39) is titled *\btnAnime: Animation using Form Field Buttons with AeB Pro* can be found at the AcroTeX Blog web site.

To create a button anima, follow these steps:

1. Create your animation by placing one frame of your animation on one page of a PDF. If the anime has 40 frames, the PDF has 40 pages, each page contains one frame. The anime must be placed on the page exactly in the same place to avoid any noticeable shaking or trembling of the anime as it plays. (I use PSTricks to create a few of the demo animations.) Give your animation PDF some name, say, `myAnime.pdf`

2. In the preamble, place the following commands

```
\embedMultiPageImages{lastpage=36,name=myAnime,
    path=graphics/myAnime.pdf,placement=myFirstAnime}
\placeAnimeCtrlBtnFaces{btn_anime_icons1.pdf}{myFirstAnime}
\begin{docassembly}
\insertPreDocAssembly;
\executeSave();
\end{docassembly}
```

The commands `\embedMultiPageImages` and `\placeAnimeCtrlBtnFaces` are described in detail in 'Methods in support of Button Anime' on page 39. These commands embed the icons (or graphic images) in the PDF document, and associates them with the anime being created.

3. Use the `\btnAnime` command to create your animation. `\btnAnime` is described below, for now, we present an example.

```
\btnAnime{%
    fieldName=myFirstAnime,iconName=myAnime,nFrames=36,
    controls=skin3,nospeedcontrol,type=loop,
    autorun,autopause
}{72bp}{72bp}
```

The centerpiece of button anime is the \btnAnime command, which is the one that actually creates the button fields to display the animation, and the button to control the anime.

> \btnAnime{*KV-pairs*}{*width*}{*height*}

Command Description: Create a series of stacked buttons that hold and display the frames of the animation; it also creates the control buttons.

Parameter Description: The first parameter,[18] taking key-value pairs, is describe below, the *width* and *height* parameters are the width and height of the button fields to display the animation. These buttons are all the same size and stacked one on top the other.

Key-Value Pairs: The first optional parameter takes key-value pairs (*KV-pairs*).

- fieldName: The base name of the anime fields to be created. The key fieldName corresponds to the placement key of \embedMultiPageImages.

- iconName: The base name of the icon set to be used in this anime. The iconName key corresponds to the name key of \embedMultiPageImages.

- nFrames: The number of frames in this anime; must be the same number as lastpage-firstpage+1 as declared in \embedMultiPageImages. (If the first-page key is not used, then nFrames=lastpage.)

- type: This is a choice key that takes any of three values: loop, palindrome, or stopatboundary. The latter being the default. For loop, the goes through the stack of frames from 1 to nFrames, then repeats 1 to nFrames, and so on until the anime is paused. For palindrome, the anime plays in the order 1 to nFrames, nFrames-1 to 1, then repeat. This anime continues until the paused. For stopatboundary, the anima pauses when the frame reaches nFrames for forward play, and pauses at frame 1 for backward play.

- poster: This is a choice key that takes any of three values: first, last, and none; the default is first.

- speed: When the anime is played, the value of speed is the initial or default speed of the animation. The speed is measured in *milliseconds*. There are controls for changing the speed dynamically. When the speed key is not listed, the default speed is 200 milliseconds.

- autorun: Determines whether the anime plays when the page is either open or becomes visible; see autoplayevent below. The default is autorun=false. Listing autorun in the list of options is the same as autorun=true.

---

[18]In version 1.1 (dated 07/10/10) the first parameter was optional, since there are required keys in this parameter, I've changed this first parameter to required.

- `autopause`: Determines whether the anime pauses when the page is either closed or becomes invisible; see `autopauseevent` below for more detail. The default is `autopause=false`. Listing `autopause` in the list of options is the same as `autopause=true`.

- `autoplayevent`: This is a choice key that takes one of two values: `pageopen` or `pagevisible`. The distinction between these two only becomes significant when the user is viewing pages continuously. The default is `pageopen`.

- `autopauseevent`: This is a choice key that takes one of two values: `pageclose` or `pageinvisible`. The distinction between these two only becomes significant when the user is viewing pages continuously. The default is `pageclose`.

The following keys concerning the buttons that control the anime.

- `ctrlwidth`: The common width of the various control buttons, the default is `18bp`.

- `ctrlheight`: The common height of the various control buttons, the default is `9bp`.

Note: The next two keys `ctrlbdrycolor` and `ctrlbdrycolor`, are needed to get the space between the buttons and the space between the rows correct. These two parameters, under different names, can be set through `\btnAnimeCtrlPresets`, but it is not recommended that you use this command to set the border color or size, use the following two keys are part of the key-value list.

- `ctrlbdrycolor`: Three numbers representing color in the RGB space. This color is used as the color of the boundary line for the button. The default color is transparent, obtained by simply listing `ctrlbdrycolor` with no value.

- `ctrlbdrywidth`: A choice key determining the width of the boundary line (or stroke). The choices are `thin`, `medium`, and `thick`, and correspond to a boundary line of width `1bp`, `2bp`, and `3bp`, respectively. The default is `thin` (`1bp`).

- `controls`: Determines the design of the set of control buttons. The following values are recognized: `none`, `skin1`, `skin2`, `skin3`, `skin4`, `skin5`, `skin6`. When `controls=none`, no controls are displayed (better use `autoplay/autopause`); skins 1–4 have various buttons included in them, `skin1` includes all buttons.

  Values of `skin5` and `skin6` are left for the author to design his/her own button layout. Without a redefinition, these skins opt to `skin1`. See the appropriate section of `aeb_pro.dtx` to see who one might create a custom layout.

  The space between control buttons is determined by `\btnanimebtnsep`, the default definition is `\newcommand{\btnanimebtnsep}{1bp}`.

  `\aep@vspacectrlsep`: The vertical space between the bottom of the anime and the control button set, `\newcommand{\aep@vspacectrlsep}{3bp}` is the default definition.

- `nospeedcontrol`: There are two buttons for increasing and decreasing the speed. (The minimal speed is 10 milliseconds, by the way.) If `nospeedcontrols` are included in the option list, then this Plus/Minus pair of buttons are not included in the set of control buttons.

- `usetworows`: If `usetworows` is in the option list, then two rows are taken to list all the buttons. The second row usually consists of the Plus/Minus buttons.

  The vertical space between two rows of buttons is `\btnanimerowsep`, the default definition is `\newcommand{\btnanimerowsep}{1bp}`.

Controlling the appearance of the Anime Fields. You can influence of the appearance of buttons that display the images by using the command `\btnAnimePresets`.

```
\btnAnimePresets{KV-pairs}
```

Parameter Description: The key values are ones associated with button form fields, as described in the eForm reference.

Controlling the appearance of the Anime Control Buttons. You can influence of the appearance of the buttons that provide the control for the anime by using the command `\btnAnimeCtrlPresets`.

```
\btnAnimeCtrlPresets{KV-pairs}
```

Parameter Description: The key values are ones associated with button form fields, as described in the eForm reference.

16.2. The `ocgAnime` Environment

AeB Pro has provided for several years a basic animation feature using layers (or, in Adobe's terminology, OCG, optional content groups). We now upgrade OCG animation up to the same level as button anime, much of the same code is used. Code for OCG anime is include when the `ocganime` option is taken.

🌐 The demo file for the material in this section, is titled *ocgAnime: Animation using OCG (Layers) with AeB Pro* can be found at the AcroTeX Blog web site.

OCG animation is available through the `ocgAnime` environment.

```
\begin{ocgAnime}{KV-pairs}
  ⟨a set of ocg frames built using \animeBld/\eBld pairs⟩
\end{ocgAnime}
```

Key-Value Pairs: The key-value pairs are the same ones described in Section 16.1 on page 54. The `iconName` key is not recognized (this is a button anime key), and `ocgAnimeName` is an alias for `fieldName`, `ocgAnimeName` being more descriptive of the base name for the OCG animation. The two keys `ocgAnimeName` and `nFrames` are required.
Below is an example of this syntax.

```
1   \begin{ocgAnime}{ocgAnimeName=sineAnime,nFrames=41,
2       type=palindrome,speed=10,controls=skin1}
3   \FPdiv{\myDelta}{\psPiTwo}{40}
4   \def\thisframe{\animeBld\psplot[linecolor=red]{0}{\xi}{sin(x)}\eBld}
5   \def\xi{0}\psset{algebraic=true}
6   \psset{llx =-12pt,lly=-12pt,urx =12pt,ury =12pt}
7   \begin{psgraph*}[arrows=->,trigLabels=true,trigLabelBase=2,
8       dx=\psPiH](0,0)(-.5,-1.5)(6.75,1.5){164pt}{70pt}%
9       \rput(4,1){$y=\sin(x)$}%
10      \animeBld\eBld
11      \multido{\i=1+1}{40}{\FPadd{\xi}{\xi}{\myDelta}\thisframe}%
12  \end{psgraph*}
13  \end{ocgAnime}
```

As with \btnAnime, \placeAnimeCtrlBtnFaces is used to import the icon appearances of the control buttons.

16.3. Moving the Control Buttons

The default location of the control buttons is below the anime. It is possible to move them elsewhere. Use the commands \animeSetup and \insertCtrlButtons for this purpose.

The demo file for the material in this section, is titled *Moving the Control Buttons for Button and OCG Animation* can be found at the AcroTeX Blog web site.

\animeSetup{*KV-pairs*}

Command Description: The argument is the key-value pairs of \btnAnime or of the ocgAnime environment. The command processes the key-value pairs.

Following the execution of \animeSetup, use \insertCtrlButtons to layout the buttons according to the values specified by the key-value pairs. An example follows.

```
1   \begin{minipage}[c]{190pt}\centering
2   \begin{ocgAnime}{ocgAnimeName=sineAnime,nFrames=41,
3       type=palindrome,speed=10,controls=none}
4   \FPdiv{\myDelta}{\psPiTwo}{40}
5   \def\thisframe{\animeBld\psplot[linecolor=red]{0}{\xi}{sin(x)}\eBld}
6   \def\xi{0}\psset{algebraic=true}
7   \psset{llx =-12pt,lly=0pt,urx=12pt,ury=12pt}
8   \begin{psgraph*}[arrows=->,trigLabels=true,trigLabelBase=2,
9       dx=\psPiH](0,0)(-.5,-1.5)(6.75,1.5){164pt}{70pt}%
10      \rput(4,1){$y=\sin(x)$}\animeBld\eBld % first (empty) frame
11      \multido{\i=1+1}{40}{\FPadd{\xi}{\xi}{\myDelta}\thisframe}%
12  \end{psgraph*}
13  \end{ocgAnime}
14  \end{minipage}\quad{\animeSetup{ocgAnimeName=sineAnime,nFrames=41,
15      type=palindrome,speed=10,controls=skin3,usetworows}%
16      \insertCtrlButtons}
```

Comments: The key-value pairs passed in line (2) are minimal, with `controls=none`. After closing the `minipage` environment in line (14), we begin with a left-brace ({) to enclose `\animeSetup` and `\insertCtrlButtons` in a group so all changes in the parameters are local. In lines (13)–(16), we execute `\animeSetup` with our selected options, and follow this with `\insertCtrlButtons` and we are done!

`\insertCtrlButtons` itself expands to a `\parbox`, `\insertCtrlButtons` has an optional parameter that is passed to the optional parameter of the underlying `\parbox`, permissible values are `c` (the default), `b`, and `t`.

Note that if you want to use `autorun` and `autoresume`, these parameters must be passed in the parameter set of `ocgAnime`, not from the parameter set of `\animeSetup`

Rather just enclosing `\animeSetup` and `\insertCtrlButtons` in grouping braces, you could also use a `\parbox`, and add, perhaps, a caption.

In theory, the control button can be placed anywhere on the page, above the anime, below it, to the left or to the right of it.

🌐 The AcroTeX Blog post titled *Custom Designing Anime Control Button Layout* shows how to design your own button layout, and how to define your own "skin."

One last command, `\insertCtrlButtons` is normally expanded at the end of the `\btnAnime` or `ocgAnime` environments. It appears in the form

    `\ctrlButtonsWrapper{\insertCtrlButtons}`

🌐 The command `\ctrlButtonsWrapper` can be redefined to create special effects, as illustrated in the AcroTeX Blog post titled *Some Comments on Anime Button Layouts.*

References

[1] "**exec**JS: 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). See page 31.

[2] JavaScript for Acrobat® API Reference, Adobe® Acrobat® SDK, Version 8.0., Adobe Systems, Inc., 2006. See pages 25, 27, 31, and 34.

http://www.adobe.com/go/acrobat_developer

[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. See page 4.

http://www.adobe.com/go/acrobat_developer

[5] PDF Reference, Version 1.7., Adobe Systems, Inc., 2006. See page 27.

http://www.adobe.com/go/acrobat_developer

[6] D. P. Story, *AcroTeX eDucation System Tools: LaTeX for interactive PDF documents*, in preparation. See page 4.

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