AcroTEX.Net

Support for Links in AeB/eForms

D. P. Story

AeB Blog January 2, 2010

http://www.acrotex.net
Published: January 2, 2010

Table of Contents

1. Introduction

In this blog, we discuss the support for link annotations, through the command `\setLink`. The syntax for `\setLink` is

```
\setLink[<key-values>]{<link_text>}
```

The `<link_text>` is the text (or any LaTeX object) that is to be made into a link; it is through the optional first argument with its `<key-values>` that you define the appearance and the actions of the link. For users of AeB, you know that there are two key-value systems that I've implemented over the years:

1. The first key-value system that I developed uses key-values of the form `\<key>{value}`, for example, the link Click on me!
   ```
   \setLink[\linktxtcolor{blue}\Color{blue}
       \W1\S{U}\H{P}]{Click on me!}
   ```
   sets the link text to blue, the boundary color to blue too, sets the boundary width of the boundary to be 1 point, sets the line style of the boundary line to underline, and sets the highlight style to push. Of course, if there are no optional arguments, the link appearance has a default Click on me! The color of the link is called `webgreen` and is defined in the web package. More details on the appearance are given in the section 2.
2. The other key-value pair system that `\setLink` understands (as well as all Acrobat form constructs of eforms), is the key-value pair system based on xkeyval. To use the "user interface" key-value system, you must use the `useui` option of eforms. (This option can also be passed to eforms as an option of the web package.) The fancy link created above can be created as well: Click on me!
   ```
   \setLink[\ui{border=visible,linktxtcolor=blue,
       bordercolor=blue,linestyle=underlined,
       linewidth=thin,highlight=push}]{Click on me!}
   ```
   Notice that the key-values are enclosed in `\ui{...}`, the `\ui` command is the one that is used to signal that the key values are those of the xkeyval style. Again, additional details are found in section 2.

The original key-value system, one which I use, is much more compact and flexible, but requires a knowledge of the rather arcane keys; the ui system uses a more lengthy style, but the keys are more suggestive of their designed purpose.

2. The Appearance of a link

The appearance of a link can be accessed through the user interface, for anyone who has the Acrobat application. The appearance tab of the Link Properties dialog box appears in Figure 1.

The properties of the link are Link Type, Highlight Style, Line Thickness, Line Style, and Color. In addition to these, we include the coloring of the link text. These are key-value pairs corresponding to each of these properties. We itemize these properties along with the corresponding key-value pairs of eforms.

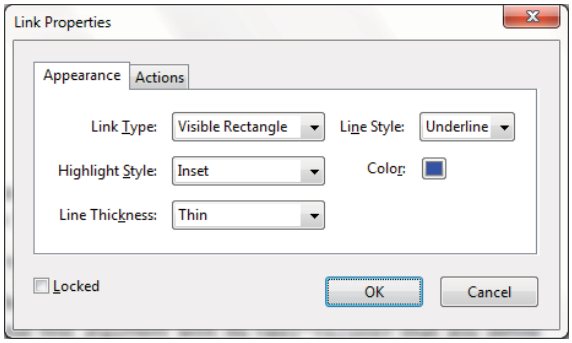


Figure 1: Appearance Tab

- Link Type: The user interface offers two possibilities: Visible Rectangle, and Invisible Rectangle.
  - Original Key-Value System: The key to make the rectangle visible is `\W`. The value of this key is a nonnegative integer: 0 (Invisible Rectangle), 1 (Thin Border), 2 (Medium Border), and 3 (Thick Border). When the value of `\W` is greater than zero, the rectangle is visible. For example, `\W{0}` is an invisible rectangle (this is the same as `\W{}`), while `\W{1}` gives a rectangle with 1 point width.
  - The xkeyval System: The `border` key takes one of two values `visible` or `invisible`. `border=invisible` sets the border width to 0, while `border=visible` sets border with to thin (`\W{1}`).
- Highlight Style: The user interface offers four possibilities: None, Invert, Outline, Inset.
  - Original Key-Value System: The `\H` key controls the highlight style. Possible values are empty (None), I (Invert), O (Outline), and P (Inset). For example `\H{}` corresponds to None (in the user interface, there is no highlighting when the link is clicked). `\H{O}`.
  - The xkeyval System: The key here is `highlight`, it has values `none`, `invert`, `outline`, and `inset`. For example, `highlight=outline` sets the highlight style into the outline style (Click here).
- Line Thickness: The thickness of the bounding rectangle. The user interface offers Thin, Medium, and Thick. When the Link Style is set to Invisible Rectangle, this property is inactive (the border width is set to 0, `\W{0}`).
  - Original Key-Value System: The key to make the rectangle visible is `\W`. The value of this key is a nonnegative integer: 0 (Invisible Rectangle), 1 (Thin Border), 2 (Medium Border), and 3 (Thick Border). When the value of `\W` is greater than zero, the rectangle is visible. For example, `\W{0}` is an invisible rectangle (this is the same as `\W{}`), while `\W{1}` gives a rectangle with 1 point width.
  - The xkeyval System: The `linewidth` key is used to set the line thickness. Possible values are `thin`, `medium`, and `thick`. This key is ignored if `border=invisible`.
- Line Style: This item is inactive when the Link Style is set to Invisible Rectangle; otherwise, the choices are Solid, Dashed, And Underline.

  - Original Key-Value System: The \S key is used to set the line style, possible values are S (Solid), D (Dashed), and U (Underline). For example, \S{U} sets the line style to underline (Click me), or dashed (Click me). The code for these two are
    ```
    \setLink[\W1\S{U}]{Click me}
    \setLink[\W1\S{D}]{Click me}
    ```
    The colors do not match very well, had we set the border color to match the link color, link would have looked better.
  - The xkeyval System: The linestyle key is used to set the Line Style, possible values are `solid`, `dashed`, and `underlined`.

- Color: The border color is set through a color palette in the user interface. The color comes from the RGB color space.
  - Original Key-Value System: The `\Color` key is used to set the line style, the value of this key is list of three numbers between 0 and 1 that corresponds to the RBG color space, for example `\Color{1 0 0}` sets the border to red. Note the numbers are delimited by a space, *not a comma*. If the xcolor package is used, you can enter a named color, for example, `\Color{red}`.
  - The xkeyval System: The `bordercolor` is used to set the (Border) Color. As in the original key-value scheme, the value is the same as described in the original key-value scheme, for example, `bordercolor=0 1 0` or `bordercolor=green` (if xcolor is used).

  The next two keys are not available through the user interface, the first one is a LaTeX property (link color), the second one is in the PDF Specification.

- Link Color: Set the color of the link through the optional parameters of `\setLink`. Color is used to color the links when the `colorlinks` option of hyperref is used. The web package automatically uses the `colorlinks` option.
  - Original Key-Value System: The key `\linktxtcolor` takes as its value a named color (in the (x)color scheme). The default is whatever the hyperref sets the link color as, though the web package redefines the link colors, e.g., `\linktxtcolor{blue}`.
  - The xkeyval System: Use the key `linktxtcolor` to set the color of the link text. For example `linktxtcolor=blue`.

- Dash Array: When Line Style is set to Dashed (`\S{D}` or `linestyle=dashed`), and the Link Type is set to Visible Rectangle (`\W{1}` or greater, or `border=visible`), a dashed line is drawn as follows: the view repeatedly draws 3-point dashes followed by 3-point gaps. This is the default dashed border: Dash it all!
  ```
  \setLink[\linktxtcolor{black}\Color{red}\W{1}\S{D}
      ]{Dash it all!}
  ```
  The design of the dash array can be customized. The format of the dash array is described in the section titled Dash Line Pattern, page 217, of version 1.7 of the PDF Specification; examples a dash array of `[4]` defines a 4-point dash followed by 4-point gap, `[4 2]` defines a 4-point dash followed by a 2-point gap.
  - Original Key-Value System: Use the `\D` key to set the dash array, for example, `\D{6 2}`. Dash it all! versus the default Dash it all!, can you tell the difference?

```
\setLink[\linktxtcolor{black}\Color{red}
    \W{2}\S{D}\D{6 2}]{Dash it all!}
\setLink[\linktxtcolor{black}\Color{red}
    \W{2}\S{D}]{Dash it all!}
```

I've increased the border width to \W{2} for greater visibility of the dashed border.

– The xkeyval System: The access to the dash array is through the dasharray key, for example, Dash it all! This example is the same as the one given in the original key-value paragraph, the code is,

```
\setLink[\ui{linktxtcolor=black,bordercolor=red,
    border=visible,linewidth=medium,
    linestyle=dashed,dasharray=6 2}]{Dash it all!}
```

If you have a combination of properties you want to use for your links, rather than copying and pasting a (possibly, long) list of properties, you can define these properties as *presets*. After defining preset properties, use the \presets (original kv-system) key or the presets key (xkeyval system) to apply them.

- Original key-value system: We can define our favorite key-values like so,

```
\newcommand{\myPresets}{\linktxtcolor{blue}
    \Color{blue}\W1\S{U}\H{P}}
```

  Create a link using \myPresets as the property of \presets, like this one, Click on me!. The code is

```
\setLink[\presets{\myPresets}]{Click on me!}
```

- The xkeyval system: Similarly, you can define your favorite properties

```
\newcommand{\myUIPresets}{border=visible,linktxtcolor=blue,
    bordercolor=blue,linestyle=underlined,
    linewidth=thin,highlight=push}
```

  and uses this command as the value of the presets key, the link Click on me! was created using the code

```
\setLink[\ui{presets=\myUIPresets}]{Click on me!}
```

3. The Actions of a link

A link is not very interesting unless there is an associated action. The Table 1 lists some common actions for links. When uses \setLink, actions are introduced into the link using the \A key.

3.1. GoTo Actions

GoTo actions are supported by \setLink of the eforms package, and by the \hyperref and \hyperlink commands of the hyperref package. When we execute GoTo, we jump from the current location of the link to either a *named destination* or a *page number* within the *same document*.

Action	Description	Examples
GoTo	Go to a destination in the current document	Page 6
GoToR	Go to a destination in another document	Page 8
Launch	Launch an application, usually to open a file	Page 12
URI	Resolve a uniform resource identifier	Page 11
Named	Execute an action predefined by the viewer	Page 14
JavaScript	Execute a JavaScript script (PDF 1.3)	Page 15

Table 1: Partial List of Actions

• Jumping to Named Destinations

hyperref has two methods for creating named destinations:

1. The hyperref package creates destinations that support the LaTeX cross-referencing system. The hyperref package creates internal named destinations for sections, subsections, lists, equations, and so on. The `\label` command is the user interface to the internal named destinations assigned to the various LaTeX markup constructs. The creation of these destinations are automatic, and are accessible through the names assigned by the user using `\label`. The commands `\hyperref`, `\ref`, `\pageref` (and others), and `\setLink` can be used to jump to this type of destination.
2. You can assign explicit destinations using the hyperref command `\hypertarget`. The commands `\hyperlink` and `\setLink` can be used to jump to this type of destination.

In the working examples that follow, we jump to the label defined at the beginning of this section:

```
\section{The Actions of a link}\label{actions}
```

We also jump to an explicit destination defined earlier in this document

```
\hypertarget{presets}{\textit{presets}}
```

Example 1: Jumping to a destination defined by a `\label`

- Using `\setLink`: Jump to the labeled destination: GoTo!

  ```
  \setLink[\A{/S/GoTo/D(\labelRef{actions})}]{GoTo!}
  ```

  Use the `\A` key to define an action; use `/S/GoTo` to specify a `GoTo` action; use the `/D(...)` key-value pair to specify a destination. The destination is accessed through cslabelRef, a command defined in the eforms package. Here, we use the default appearance of the link.

  The same link can be set using the xkeyval system: GoTo!

  ```
  \setLink[\ui{goto={labeldest={actions}}}]{GoTo!}
  ```

  To jump to a labeled target, use the `goto` key, the value of this key itself takes key-value pairs. In this case, we set the target with `labeldest={actions}`

- Using `hyperref` commands:
  - `\hyperref`: GoTo! (`\hyperref[actions]{GoTo!}`)
  - `\ref`: Section 3 (`Section~\ref{actions}`)

- \pageref: On page 6 (page~\pageref{actions})
- \autoref: section 3 (\autoref{actions})
- \nameref: The Actions of a link (\nameref{actions})
- \Nameref: 'The Actions of a link' on page 6 (\Nameref{actions})

There are others commands created by other packages that are integrated into LaTeX cross-reference system that hyperref converts into links as well.

Example 2: Jumping to an explicit destination defined by \hypertarget

- Using \setLink: Jump to the labeled destination: GoTo!

  ```
  \setLink[\A{/S/GoTo/D(presets)}]{GoTo!}
  ```

  This is the same syntax as seen in an earlier example, except the destination string is the actual destination string presets.

  Using now the xkeyval system: GoTo!

  ```
  \setLink[\ui{goto={targetdest=presets}}]{GoTo!}
  ```

  When jumping to an explicit destination, we use the targetdest key (rather than the labeldest key).
- The \hyperlink command, defined by hyperref is used to jump to an explicit destination: GoTo! (\hyperlink{presets}{GoTo!})

• Jumping to a particular Page

The GoTo action can be used to jump to a particular page within the same document.

- Using eform's \setLink: Jump to page 2!

  ```
  \setLink[\A{/S/GoTo/D[\Page{2}\fitpage]}]{Jump to page 2!}
  ```

  The destination key /D has an array as its value (enclosed in brackets) consisting of a page number (entered using the special \Page command) followed by a view, possible values for \fitpage,\actualsize, \fitwidth, \fitvisible, and \inheritzoom. These are the view settings available through the user interface. The page number here is a base 1 number, that is, a value of one jumps to the first page of the document.
- Using hyperref's commands: The hyperref package has no command to jump to an explicit page number; however, hyperref does define *named destinations* on each page enabling you to jump to page 2 anyway. The code is

  ```
  \hyperlink{page.2}{jump to page 2}
  ```

  The example shows the format of the named page destination; this too is a base 1 numbering system. Note the use of the \hyperlink command (as opposed to the \hyperref command that is used for labeled destinations).

3.2. GoToR Actions

The GoToR action allows you to jump to a destination (named destination or explicit page number) in *another document*. This action works on a local computer file system (current file is being viewed with a conforming PDF viewer, such as Adobe Reader) and from within a browser as long as the path to the other document is relative the current document.

• Jumping to Named Destinations

We illustrate jumping to a named destination using \setLink, of the eforms package, and \hyperref and \href of the hyperref package.

Example 3: Jumping to a destination defined by a \label
The target file, aeb_link_tst.pdf, contains a section with a label attached:

```
\section{Test Section}\label{testsection}
```

We shall used this label as a target for the examples that follow. To jump to labels defined in another document that has been created by LaTeX with the hyperref package, it is necessary to use the xr-hyper package; we also included an \externaldocument command in the preamble

```
\usepackage{xr-hyper}
...
\externaldocument[lnktst:]{aeb_link_tst}
```

The \externaldocument gathers up all the labels defined in aeb_link_test.tex, prepends their names with lnktst: so there should not be any name conflicts with the current document.

- Using \setLink: GoToR!

  ```
  \setLink[\A{/S/GoToR/F(aeb_link_tst.pdf)
      /D(\labelRef{lnktst:testsection})}]{GoToR!}
  ```

  We need to get the internal named destination generated by hyperref so we use the \labelRef command as before, defined in eforms. You can see from the code above, the label testsection is referenced with the prefix lnktst: to obtain a cross-reference label of lnktst:testsection.

  Here is the same example using the user friendly scheme: GoToR!

  ```
  \setLink[\ui{goto={file=aeb_link_tst.pdf,
      labeldest=lnktst:testsection}}]{GoToR!}
  ```

  Note that we use the labeldest, \setLink automatically uses the \labelRef command internally to compute the correct target.

- Using \hyperref: Jump to a label!

  ```
  \hyperref[lnktst:testsection]{Jump to a label!}
  ```

  Or, go to 'Test Section' on page 6 (\Nameref{lnktst:testsection}).

Example 4: Jumping to an explicit destination defined by \hypertarget
The target file, aeb_link_tst.pdf, contains a page with a \hypertarget

```
\textit{\hypertarget{dest}{destination}}
```

We jump to this destination.

- Using \setLink: GoToR

  ```
  \setLink[\A{/S/GoToR/F(aeb_link_tst.pdf)/D(dest)}]{GoToR}
  ```

Because this target is not part of the LaTeX cross-referencing system, we don't prepend 'lnktst:' as we did earlier. The target name is dest, which we reference as /D(dest). Here is the same link using the user-friendly key-value system: UI GoToR!

```
\setLink[\ui{goto={file=aeb_link_tst.pdf,
    targetdest=dest}}]{UI GoToR!}
```

- Using hyperref commands: For GoToR actions that are not URLs, use \href, like so: Go there man!

```
\href{aeb_link_tst.pdf#dest}{Go there man!}
```

The destination follows the number sign (#), and is interpreted by the hyperref package to be a named destination. The \href command is a multipurpose command, it expands to a GoToR action when there is no URL (http:, https:, mailto:, file:, or run:). The situations listed in the parentheses will be covered later in this blog.

• Jumping to a particular Page

- Using \setLink: Jump to page 3 of aeb_link_tst.pdf: GoToR page 3

```
\setLink[\A{/S/GoToR/F(aeb_link_tst.pdf)
    /D[\rPage{3}\fitpage]}]{GoToR page 3}
```

In the destination array /D, the page is specified using the \rPage (remote page) command. The pages are 0-based, this command decrements the page by one, so that if you specify \rPage{3} as we have done here, the jump is to the third page. The second part of the destination array is a desired view. The choices are \fitpage, \actualsize, \fitwidth, \fitvisible, and \inheritzoom. You can also put in a custom view, if you know enough about the format. These names correspond to the possible choices in the user interface of Acrobat.

The same link using the user-friendly scheme is GoToR page 3

```
\setLink[\ui{goto={file=aeb_link_tst.pdf,
    page=3,view=fitpage}}]{GoToR page 3}
```

In the user-friendly scheme, we specify the page and view as separate keys.

- Using hyperref commands: Go there man!

```
\href{aeb_link_tst.pdf#page.3}{Go there man!}
```

Hyperref has no command that jumps to an explicit page number; however, hyperref interprets the string following the number sign (#) as a named destination. If you are jumping to a document created using the hyperref package, you can jump to the destination page.3. Hyperref defines a destination by the name of page.<n> on each page.

If you want to jump (using GoToR) to a page of a document that *was not created* using LaTeX/hyperref on the same file system or on the same web-server, use \setLink as illustrated above. For jumping to a non-LaTeX/hyperref document on a different web-server, use the URI action with \setLink, explained below.

3.3. URI Actions

A URI action is an action that jumps to a resource on the Internet, usually a PDF file or a web page.

- Using \setLink: My Home Page

```
\setLink[\A{/S/URI
    /URI(http://www.math.uakron.edu/~dpstory)}]{My Home Page}
```

  There are a couple of helper commands defined in the eforms package, \URI and \definePath. The command \URI expands to /S/URI/URI(#1), and \definePath can be used to define paths, for example, we define,

```
\definePath{\myHP}{http://www.math.uakron.edu/~dpstory}
```

  then we can say My Home Page

```
\setLink[\A{\URI{\myHP}}]{My Home Page}
```

  Using the user friendly key-value system we have My Home Page

```
\setLink[\ui{goto={url={\myHP}}}]{My Home Page}
```

  Here, within the value of goto, we specify the url key-value pair (rather than the file key value pair for a GoToR action).

- The hyperref package provides the general purpose \href for jumping to Internet resources: My Home Page

```
\href{http://www.math.uakron.edu/~dpstory}{My Home Page}
```

The above examples focus on jumping to an html page, the syntax is similar when jumping to a PDF file.

Example 5: Jumping to a PDF file
In some of the examples that follow, we use the open parameters from *PDF Open Parameters* found at the Acrobat Developer Center.

We use one of the articles from the AeB Blog.

1. **Jump to a PDF file:** Demo Deco and Demo Deco

```
\setLink[\A{\URI{\aebBlog/deco_border.pdf}}]{Demo Deco} and
\href{\aebBlog/deco_border.pdf}{Demo Deco}
```

2. Jump to a particular page: Demo Deco page 2 and Demo Deco page 2

```
\setLink[\A{\URI{%
    \aebBlog/deco_border.pdf#page=2}}]{Demo Deco page 2}
and \href{\aebBlog/deco_border.pdf#page=2}{Demo Deco page 2}
```

3. The document deco_border.pdf has attachments and is set to display the attachments panel. To view the document without opening the attachments panel we do this Demo Deco page 2 and Demo Deco page 2

```
\setLink[\A{\URI{%
    \aebBlog/deco_border.pdf#page=2&pagemode=none}}]{Demo Deco page 2}
and \href{\aebBlog/deco_border.pdf#page=2&pagemode=none}{Demo Deco page 2}
```

4. **Jump to a named destination:** Partial Credit and with hyperref Partial Credit.

```
\setLink[\A{\URI{\aebBlog/%
    jtxttst_pc.pdf#nameddest=paraRespBoxTxt}}]{Partial Credit}
\href{\aebBlog/%
    jtxttst_pc.pdf#nameddest=paraRespBoxTxt}{Partial Credit}
```

The destination name is `paraRespBoxTxt`, and was defined by the `\hypertarget` command. This document has an open attachments panel, which can be removed using `pagemode=none`.

I haven't illustrated the user-friendly notation: Partial Credit

```
\setLink[\ui{goto={url={\aebBlog/%
    jtxttst_pc.pdf#nameddest=paraRespBoxTxt&pagemode=none}}}]
    {Partial Credit}
```

3.4. Launch Actions

A *launch action* launches an application, or opens or prints a document.

Example 6: Launching an application to open a document.
When you pass a document to the `Lanuch` action that has an extension with an associated application, the application is launched to view the document.

In the example below, if you have an editor that launches when you double-click on a file with an extension of `.tex`, that editor will launch when you click on any of the following links.

- Using `\setLink`: View Source

```
\setLink[\A{/S/Launch/F(aeb_links.tex)}]{View Source}
```

- `\setLink` (UI scheme): View Source

```
\setLink[\ui{launch={file={aeb_links.tex}}}]{View Source}
```

- Using `\href`: View Source

```
\href{run:aeb_links.tex}{View Source}
```

Note the use of '`run:`' to signal to `\href` that the file is to be run (or launched).

- When the Launch action is used in this way, the path to the file need only be relative to the current directory. Here is an example of opening a spreadsheet:

```
\setLink[\A{/S/Launch
    /F(../../class09/ca_roster.xls)}]{Open a spreadsheet!}
```

- When you launch a PDF file, there is also another key to open this PDF in a new window:

  Launch, no new window: View a PDF

```
\setLink[\A{/S/Launch/F(aeb_link_tst.pdf)}]{View a PDF}
```

  Launch, new window: View a PDF

```
\setLink[\A{/S/Launch/F(aeb_link_tst.pdf)
    /NewWindow true}]{View a PDF}
```

  Launch, new window user-friendly, View a PDF

```
\setLink[\ui{launch={file=aeb_link_tst.pdf,
    open=new}}]{View a PDF}
```

The key open has three possible values: `userpref` (the default), `new`, `existing`.

Example 7: Using the Win parameters.
The *PDF Specification* allows for a `Win` dictionary that allows you to pass command line parameters to the application. Through experimentation, the paths used within the `Win` dictionary need to be absolute. For this reason, the use of this dictionary seems rather limited, in the sense that the PDF cannot really be distributed to another computer system, unless that computer has the same directory system; yet, if you developing a "desktop application," one that will be used only on your computer, the `Win` parameters may be useful.

In the examples that follow, we use the command `\myEPS`, defined in the preamble as

```
\definePath{\myEPS}{"C:/Users/DPStory/Documents/TeX Stuff/papersize/jimAlex.eps"}
```

Note that the path is enclosed in double quotes.

- Distill the file:

  ```
  \setLink[\A{/S/Launch
      /Win<</F(acrodist.exe)/P(\myEPS)>>}]{Distill a file!}
  ```

  When the the `Win` dictionary is present the the `F` key following `/S/Launch` is optional, we leave it out. The dictionary here shows two keys, `F` (the application to use) and `P` (the parameters). We use acrodist.exe as the application, and supply the absolute path to an EPS file to distill for the value of the `P` key.

  As a convenience, I've define a `\Win` command that expands to `/Win<<#1>>`, thus, the above example can be rewritten,

  ```
  \setLink[\A{/S/Launch
      \Win{/F(acrodist.exe)/P(\myEPS)}}]{Distill a file!}
  ```

  The user-friendly notation is

  ```
  \setLink[\ui{launch={winParams={file={acrodist.exe},
      params={\myEPS}}}}]{Distill a file!}
  ```

  For the user-friendly version, we use the key `winParams` to set the contents of the Win dictionary. Specify the `file` and the `params` keys (which populate the PDF keys `F` and `P`).

- Passing command line switches:

  ```
  \setLink[\A{/S/Launch\Win{/F(acrodist.exe)
      /P(-F \myEPS)}}]{Distill a file!}
  \setLink[\ui{launch={winParams={file={acrodist.exe},
      params={-F \myEPS}}}}]{Distill a file!}
  ```

  Placing command line switches, `-F` in this example, within the `P` key (the `params` key) seems to work.

- **Printing a file.** We print a specific PDF file using an absolute path.

  ```
  \setLink[\A{/S/Launch/Win <<
      /F ("C:/Users/DPStory/Documents/TeXStuff/myDoc.pdf") /O (print)
  >>}]{Print a file!}
  ```

The O key is used with a value of print (a string). Values of the O key are open and print. If the path leads to a folder rather than a file, the /O (open) (the default) opens that folder in the file system (Windows explorer). Windows also recognizes /O (explore).

User-friendly version:

```
\setLink[\ui{launch={winParams={%
        file={"C:/Users/DPStory/Documents/TeXStuff/myDoc.pdf"},
        open=print}}}]{Print a file!}
```

- **Open or Explore a folder.** Open or Explore a folder.

```
\setLink[\A{/S/Launch/Win <<
    /F ("C:/Users/DPStory/Documents/TeXStuff/") /O (open)
>>}]{Open a Folder!}
\setLink[\A{/S/Launch/Win <<
    /F ("C:/Users/DPStory/Documents/TeXStuff/") /O (explore)
>>}]{Explore a Folder!}
```

Again, these links—those with the Win (winParams) key—don't seem very useful unless it is for a fixed file system.

3.5. Named Actions

There is a type of action call named actions. There are four names given in the PDF Reference: NextPage, PrevPage, FirstPage, LastPage. An example of this type of actions would be Previous Page.

```
\setLink[\A{/S/Named/N/PrevPage}]{Previous Page}
```

The hyperref package version of this is Previous Page uses the \Acrobatmenu command:

```
\Acrobatmenu{PrevPage}{Previous Page}
```

eforms defines a helper command, \Named, which expands to /S/Name/N/#1. The \setLink example can be written Previous Page (\setLink[\A{\Named{PrevPage}}]{Previous Page}).

In theory, any menu item can be executed as a named actions; there are several factors to be taken into consideration: (1) Not all menu items available to Acrobat are listed on the menu bar of Adobe Reader, when choosing a name event to use, you should decide if the application executing the named action supports that action; (2) In recent versions, starting with version 7, there have been security restrictions on the execution of menu items, the so-called "white list." Only named actions listed on the white list are allowed to execute. The white list for version 8.0 is

Named Actions on Whitelist

AcroSendMail:SendMail	LastPage	ShowHideToolbarCommenting
ActualSize	NextPage	ShowHideToolbarData
AddFileAttachment	OneColumn	ShowHideToolbarEdit
BookmarkShowLocation	OpenOrganizer	ShowHideToolbarEditing
Close	PageSetup	ShowHideToolbarFile
CropPages	PrevPage	ShowHideToolbarFind

Named Actions on Whitelist

DeletePages	Print	ShowHideToolbarForms
ExtractPages	PropertyToolbar	ShowHideToolbarMeasuring
Find	Quit	ShowHideToolbarNavigation
FindCurrentBookmark	ReplacePages	ShowHideToolbarPageDisplay
FindSearch	RotatePages	ShowHideToolbarPrintProduction
FirstPage	SaveAs	ShowHideToolbarRedaction
FitHeight	Scan	ShowHideToolbarTasks
FitPage	ShowHideAnnotManager	ShowHideToolbarTypewriter
FitVisible	ShowHideArticles	SinglePage
FitWidth	ShowHideBookmarks	Spelling
FullScreen	ShowHideFields	Spelling:Check
GeneralInfo	ShowHideFileAttachment	TwoColumns
GeneralPrefs	ShowHideModelTree	TwoPages
GoBack	ShowHideOptCont	Web2PDF:OpenURL
GoForward	ShowHideSignatures	ZoomTo
GoToPage	ShowHideThumbnails	ZoomViewIn
InsertPages	ShowHideToolbarBasicTools	ZoomViewOut

Many of these names are only recognized in Acrobat, others are available for Adobe Reader; for example, the following will probably work for Reader (I haven't tested all of them): Close, FirstPage, FitHeight, FitPage, FitVisible, FitWidth, FullScreen, GoBack, GoForward, GoToPage, LastPage, NextPage, OneColumn, PrevPage, Quit, SinglePage, TwoColumns, TwoPages, ZoomTo, ZoomViewOut.

Well, let me write a loop to test all these named actions.

Close, FirstPage, FitHeight, FitPage, FitVisible, FitWidth, FullScreen, GoBack, GoForward, GoToPage, LastPage, NextPage, OneColumn, PrevPage, SinglePage, TwoColumns, TwoPages, ZoomTo, ZoomViewIn, ZoomViewOut, Quit, test these using Adobe Reader.

3.6. JavaScript Actions

The JavaScript action is an important type of action; there was an explosion of Acrobat JavaScript API that occurred with the introduction of Acrobat 5, since then the Acrobat developers have supplied extensive API with each new feature. In recent years, however, the developers have placed security restrictions on various risky JavaScript methods. The *JavaScript for Acrobat API Reference*[1] contains a listing of Acrobat JavaScript API along with security notations; pay attention to these security restrictions.

Example 8: Demonstrating JavaScript actions for link.

- Using `\setLink`: JS Action

```
\setLink[\A{/S/JavaScript/JS(app.alert("\myMsg"))}]{JS Action}
```

  where, I have defined the macro `\myMsg` as

```
\newcommand{\myMsg}{AcroTeX rocks the world!}
```

[1] Open the navigation panel using the control in the upper left corner of the page.

There is a convenience command, \JS, that expands to /S/JavaScript/JS(#1), the above link may be rewritten as

```
\setLink[\A{\JS{app.alert("\myMsg")}}]{JS Action}
```

This is the notation that I consistently use.

Using the user-friendly syntax: JS Action

```
\setLink[\ui{js={app.alert("\myMsg")}}]{JS Action}
```

For a JavaScript action, we use the js key, the value of the key is the JavaScript code.

- Executing longer JavaScript: Longer JS Action

  In the preamble, or in the body of the document prior to the link, we use the defineJS environment, defined in the insdljs package, we write our several lines of JS.

```
\begin{defineJS}{\myJS}
console.show();
console.clear();
console.println("Let it be known throughout the world...");
console.println("that AcroTeX rocks the world!");
\end{defineJS}
```

  After this definition, we then can use these lines, like so,

```
\setLink[\A{\JS{\myJS}}]{Longer JS Action}
```

- Long JS Code: Sum integers

```
\setLink[\A{\JS{sumThruN();}}]{Sum integers}
```

  The function sumThruN() is defined at the document level. The following environment appears in the preamble of this document.

```
\begin{insDLJS}[sumThruN]{sum}{Sum Through N}
function sumThruN ()
{
    var n = app.response("Enter an positive integer","AcroTeX");
    if ( n == null ) return;
    if (isFinite(n) && n > 0) {
        n = parseInt(n);
        var sum=n*(n+1)/2;
        app.alert("The sum of the first "+n
            +" integers is "+sum+".");
    } else
        app.alert("Input not a positive integer");
}
\end{insDLJS}
```

  The insDLJS environment is defined in the insdljs package, part of AeB.

4. Multiple Actions

With JavaScript, multiple actions can easily be written, with each line of code; however, to combine several of the actions described in Sections 3.1–3.6 into one action, the Next key may be used. For example, suppose we wished to jump to page 4, then place an alert box on the page: Go there!

```
\setLink[\A{/S/GoTo/D[\Page{4}\fitpage]
/Next<<\JS{app.alert("Welcome to page 4 of the
    this document")}>>}]{Go there!}
```

The value of the Next key is a dictionary, nested within the action dictionary. We can abbreviate this construct using the \Next command (defined in eforms) that expands to /Next<<#1>>. Thus, we can Go there!

```
\setLink[\A{/S/GoTo/D[\Page{4}\fitpage]
\Next{\JS{app.alert("Welcome to page 4 of the
    this document")}}}]{Go there!}
```

A Next key has not been implemented in the user-friendly key-value scheme.

The same code does not work—outside of a browser—when jumping to another document, by the way. Go there!

```
\setLink[\A{/S/GoToR/F(aeb_link_tst.pdf)/D[\rPage{4}\fitpage]
\Next{\JS{app.alert("Welcome to page 4 of the
    target document")}}}]{Go there!}
```

We jump to the document, but the alert does not sound. On exiting this document (the document is closed) any executing JavaScript are stopped. Let's try for a work around: Go there!

```
\setLink[\A{/S/GoToR/F(aeb_link_tst.pdf)/D[\rPage{4}\fitpage]
    /NewWindow true\Next{\JS{app.alert("Welcome to page 4 of the
    target document")}}}]{Go there!}
```

Here, we open a *new window*, so this document is still open and the JavaScript initiated in this window can run.

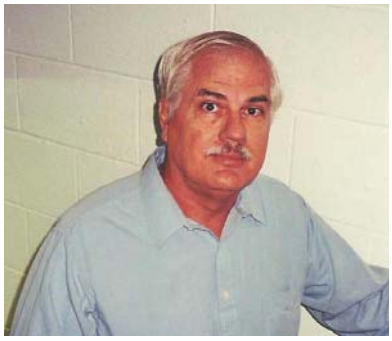
5. Multiline Links

The application pdflatex.exe can produce "multi-line links," if a link wraps around to the next line, the link is broken, then is started again on the next line. So if your link text extends over two lines, you have, in fact two links.

Beginning with Acrobat 7, the notion of a multi-line link was introduced, for this type of link, the link wraps around to the next line. When you click anywhere on the link text, the whole link text is highlighted. The aeb_mlink supports the creation of these multi-line links; the package requires the use of Adobe Distiller (version 7 or later) to create the PDF file from a Postscript file (created by dvips or dvipsone).

```
\setLink[\mlLink{true}\linktxtcolor{red}
    \A{\JS{app.alert("Multi-line links are pretty cool,
        but require the user to view them using Adobe
        Reader 7 or later.")}}]{Adobe Distiller (version 7 or later)
to create the PDF file from a Postscript file (created by dvips
or dvipsone)}
```

The \setLink command can be used (eforms 2009/12/24 or later), but to switch over to the multi-link mode, use the key-value pair \mlLink{true} in the option list. You can also use the special command \mlsetLink (instead of \setLink) which goes into multi-line mode directly. The aeb_mlink package is part of the AeB Pro family of fine LaTeX software and may be found at the home page of AeB Pro.



Well, that's pretty much it for now, I simply must get back to my retirement. 𝔇𝔖

```
\begin{center}
\begin{minipage}{2in}\centering
    \setLink[\ui{js={app.alert("So long, until next time, adios!\\n\\ndps")},
        border=visible,bordercolor=red}
        ]{\includegraphics[width=2in]{graphics/dpsweb}}\par
        Well, that's pretty much it for now, I simply must get
        back to my retirement. \dps
\end{minipage}
\end{center}
```