AcroTEX.Net

The fitr Package
Defining and Jumping to a Rectangular Destination

D. P. Story

Prepared: June 29, 2012

www.acrotex.net
Version 1.0

Table of Contents

1. Introduction

This package is an implementation of the **FitR** view-type destination. The *PDF Reference* describes **FitR** as,

> Display the page designated by page, with its contents magnified just enough to fit the rectangle specified by the coordinates *left*, *bottom*, *right*, and *top* entirely within the window both horizontally and vertically.

The package supports the dvips, dvipsone, and pdftex, luatex, dvipdfm, dvipdfmx, and xetex applications, the first two assume that **Adobe Distiller** is the PDF creator.

The only required packages are the eforms package (dated 2012/06/20 or later), which is part of the **AeB Bundle**, and collectbox by Martin Scharrer, more on this package later, and the ubiquitous xcolor.[1]

The package was developed in response to a user of the AeB Bundle who was interested in developing documents for students with low vision; the idea is to magnify regions of the document so the student can read more comfortably. The demonstration files are `fitr_demo.tex` which illustrates the package and some special methods for people with low vision, and `fitr_minimal.tex`, which is the same demo file with any and extra package stripped out.

2. The Preamble and Package Options

The minimal preamble for this package is

```
\usepackage[⟨driver⟩,⟨options⟩]{fitr}
```

The hyperref package is brought in through the eforms package. Optionally, fitr can be used with other members of AeB (web and exerquiz, for example).

Another package requirement is collectbox by Martin Scharrer; quoting from the abstract of the documentation,

> This package provides macros to collect and process an macro argument (i.e. something which looks like a macro argument) as horizontal box instead as a real macro argument. These "arguments" will be stored like when using `\savebox`, `\sbox` or the `lrbox` environment and allow verbatim or other special code. Instead of explicit braces also implicit braces in the form of `\bgroup` and `\egroup` are supported....

The `\collectbox` command is used to collect the second argument of `\jdRect`, see the discussion of `\jdRect` in Section 3. As a result, the second argument may contain verbatim text in it. Very cool.

The package has ten options: six driver options and four viewing options.

[1] The eforms package itself brings in other packages, including hyperref and insdljs.

- **Driver Options:** These are `dvips` (the default), `dvipsone`, and `pdftex` (which includes the use of lualatex), `dvipdfm`, `dvipdfmx`, and `xetex`. If you specify one of the first two, it is assumed that you are using **Adobe Distiller** as your PDF creator.

  The fitr package checks whether the web package is loaded, if so, its uses the driver used by web; otherwise fitr auto-detects for pdftex and xetex. If no driver is passed, and neither pdftex nor xetex are detected, then dvips is the default driver.

- **Viewing Options:** When you specify `preview`, the bounding boxes of the buttons are shown in the dvi-previewer (or the PDF document); you can turn off this preview by specifying `!preview` (or removing `preview` entirely from the option list). The other option type is `viewMagWin`, when this option the viewing window, a rectangular region, becomes visible in the dvi-previewer (or in the PDF document); specifying `!viewMagWin` turns off this type of preview.

  The effects of the viewing options will be illustrated later in this document, see Example 4.1 on page 7.

3. The one and only command

The fitr has only one command, `\jdRect`, but there are two forms of usage. `\jsRect` optionally creates a push button or link, and optionally creates a viewing window. The term *viewing window* refers to a rectangular region that is created by the **FitR** destination viewing specification, see **Table 8.2 Destination syntax** of the *PDF Reference*, version 1.7. A *named destination* is created and is associated with the viewing window. When we jump to a viewing window, this window is magnified to the largest extent possible. For example, click on the either of the two displayed forms of the syntax for `\jdRect`; after jumping to the viewing window, click on the same display to return to the previous view.

There are two versions of `\jdRect`, the command itself, and a * version, `\jdRect*`. The syntax follows, along with the expected parameters.

`\jdRect[`⟨*key-values*⟩`]`

The above version is used to overlay a region with a button and view window. No content is specified, but is defined by specifying the `width` and `height`; it can be positioned using `shift` and `lift`.

There is a *-version as well:

`\jdRect*[`⟨*key-values*⟩`]{`⟨*content*⟩`}`

The second parameter ⟨*content*⟩ is required when the * is present. This version is meant to enclose ⟨*content*⟩ within the button and view window. The `width` and `height`

keys are ignored, but `shift` and `lift` are obeyed (though you may `shift` or `lift` the button/view window away from the content).

Before illustrating the `\jdRect` command, we first discuss its key-value pairs.

- `lift=`⟨*length*⟩: This key-value lifts (raises) the button/viewing window up (or down); for example, `lift=15pt` (or `lift=-15pt`). The default is a lift of `0pt`. See Example 4.2 on page 7.
- `shift=`⟨*length*⟩: The amount of horizontal shift; positive to the right, negative to the left. For example, `shift=-1in` shifts the button/viewing window 1 inch to the left. The default is `0pt`. See Example 4.2 on page 7.
- `width=`⟨*length*⟩: When using `\jdRect`—as opposed to `\jdRect*`—, the width of the button and viewing window is determined by the `width` key. For example `width=1in` creates a button/viewing window that is 1 inch wide. The value of this key is ignored when the * form of the `\jdRect` is used. The default value is `0pt`. This key is required when * is not present. See Example 4.2 on page 7.
- `height=`⟨*length*⟩: Similar comments here as was made for the `width` key. This key is 0used to set the height of the button/viewing window. The default is 0pt. It is required when * is not present. See Example 4.2 on page 7.
- `ref=t|c|b`: The `ref` key-value pair determines the reference point of the button/viewing window. Permissible values are `t` top (the default), `c` center, and `b` bottom. This key is only obeyed with the `\jdRect*` form of the command; otherwise, a reference point of `b` is used.
- `adddestw=`⟨*length*⟩: The default is for the viewing window to have the same dimensions as the underlying button. The `adddestw` key-value pair is used to widen the viewing window; `adddestw=.2in` widens the window by `.2in` on the left and `.2in` on the right. See Figure 1, page 5.
- `adddesth=`⟨*length*⟩: Similar to `adddestw` but for height. The `adddesth` key-value pair is used to increase the height the viewing window; `adddesth=.2in` increases the height of the window by `.2in` on the top and `.2in` in the bottom. See Figure 1, page 5.

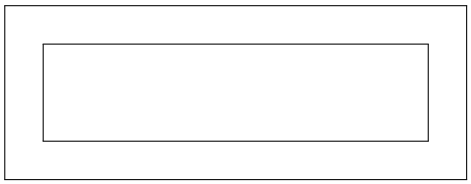

Figure 1: Button and Viewing Window
`width=2in,height=.5in,adddestw=.2in,adddesth=.2in`

- button=true|false: button is a Boolean switch. If true (the default), \jdRect creates a push button. When the user pushes the button, the viewer zooms in to the view window. Clicking the same region again restores the previous view.

  When button is false, the button is not created, but the viewing window is still created. You can then jump to the viewing window with a separate link or button. When button=false, use the dest key to assigned a numbed destination to viewing window. (fitr automatically creates the definition names internally, they are used by the buttons. If no button is created, name the destination so your know its name and can reference it in link that jumps to that viewing area.)

- link=jmp|restore If link has a value, then fitr puts button=false. The link key is used to create jumps or restore actions to or from a viewing window. When link=jmp a jump action is created, the jump will be to the value of the dest key. If this is a pure link that jumps to another viewing window, then use the nodest key as well; no viewing window will be created around the link, as it is unlikely you'll want to jump to a link.

  For example click on the link **Carl Runge** and jump to the picture of Runge in the margin. Click on the picture of Runge and return to the previous view.


Carl Runge

  The jump to the picture from the text "Carl Runge" is as follows:

```
\jdRect*[nodest,link=jmp,dest=rungePic,
adddestw=10bp,adddesth=10bp]{Carl Runge}
```

  The important options are nodest,link=jmp,dest=rungePic; no (named) viewing window is created, we want to create a jump link here, the destination of the jump link is the destination rungePic.

  The color of the link is determined by \@linkcolor, a hyperref command that holds a named color. This can be redefined at anytime, directly using

```
\makeatletter
\def\@linkcolor{blue}
\makeatother
```

  or, if you are using the pro option with the web package, you can say,

```
\selectColors{linkColor=red}
```

  When using the web package, the default is webgreen.

  The action to restore the previous view is as follows:

```
\marginpar{\jdRect*[link=restore,dest=rungePic,
    adddestw=\marginparsep,adddesth=\marginparpush
    ]{\parbox[t]{\marginparwidth}{\RungePic\\
    \normalcolor\centering\footnotesize\textsf{Carl Runge}}}}
```

The picture is placed in the margin using \marginpar; the command \RungePic is a convenience macro that uses \includegraphics in import the picture. The important options are link=restore,dest=rungePic, this first key-value pair causes \jdRect to create a restore link, the second one says to create a viewing window with a name of rungePic, this is the destination the Carl Runge link jumps to.

- nodest: A Boolean switch whose default value is false. When nodest is used (making the switch a value of true), no viewing window is created.
- dest=⟨*name*⟩: This key is a way of explicitly naming the viewing window (the destination). The destination is normally automatically generated when button=true, this key is used with the link key, as illustrated above.
- allowFX: A Boolean switch (of sorts). The fitr allows for special effects (FX) when a viewing window is jumped to and when the view is restored. The default value of allowFX is true allow special effects if there is any defined. By saying allowFX=false, no special effects are used, even if some are defined.

  An example of special effects you say? Try clicking on the Pythagorean Theorem $a^2 + b^2 = c^2$

4. Some Examples

In this section, several examples are presented that illustrate the options of \jdRect.

Example 4.1. Illustrate Preview Rectangles. The preview and viewMagWin options just set Boolean switches. In this example, we manually gives these switches a value of true (\previewtrue\viewMagWintrue). Take a close look at the following function $f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt$, or the more general form $f(x;\mu;\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}}\, dt$ The preview rectangles are shown: For the one on the left, the dimensions of the push button and the viewing rectangle are the same; for one on the right, the dimensions of the viewing window have been increased by using adddestw=20bp,adddesth=10bp. When you jump to each of these viewing windows, you the one on the left is magnified much more than the one on the right; the larger viewing window allows the user to see some of the surrounding text. □

Example 4.2. Display Math. Displayed math presents a problem. We take the following set of equations to illustrate.

Suppose we want to classify third order Runge-Kutta type methods. Start with

$$\begin{aligned} K_1 &= hf(t_n, y_n) \\ K_2 &= hf(t_n + rh, y_n + aK_1) \\ K_3 &= hf(t_n + sh, y_n + bK_1 + cK_2) \\ K &= w_1K_1 + w_2K_2 + w_3K_3 \\ y_{n+1} &= y_n + K \end{aligned}$$

Find the system of equations satisfied by $r, s, a, b, c, w_1, w_2, w_3$ that will make the above algorithm a third order method.

The verbatim listing of this set of aligned equations is

```
1  \begin{align*}
2  \jdRect[height=1.3in,width=2.6in,lift=16pt,shift=-15pt,
3      adddestw=10bp,adddesth=10bp]
4  K_1 &= hf(t_n, y_n)\\
5  K_2 &= hf(t_n +r h, y_n+aK_1)\\
6  K_3 &= hf(t_n +s h, y_n+bK_1+cK_2)\\
7  K &= w_1 K_1+ w_2 K_2+ w_3 K_3\\
8  y_{n+1} &= y_n+K
9  \end{align*}
```

This is an example of \jdRect (the non-* version), so there is no second argument. In this case, we create our button dimensions using height=1.3in,width=2.6in, line (2). Note the positioning of the \jdRect command, the upper-left point of the display. We then use lift=16pt,shift=-15pt to move the button around to cover the equations, line (2); finally, we increase the dimensions of the viewing window in line (3) with adddestw=10bp,adddesth=10bp. Now, how were the values of these keys determined? By trial and error, while the preview and viewMagWin options were in effect. Below are the same equations with \previewtrue and \viewMagWintrue, locally invoked:

$$\begin{aligned} K_1 &= hf(t_n, y_n) \\ K_2 &= hf(t_n + rh, y_n + aK_1) \\ K_3 &= hf(t_n + sh, y_n + bK_1 + cK_2) \\ K &= w_1K_1 + w_2K_2 + w_3K_3 \\ y_{n+1} &= y_n + K \end{aligned}$$

The preview rectangles do not take up any TEX space, so they overlap parts of the paragraph content. When you zoom in, you'll see part of the part of the word "invoked:", as seen in the upper-left corner, at least according to the viewing window preview. Is it so?

After you've set the position of the rectangles, and after all changes have been made to the underlying content, you don't need the preview modes. □

Example 4.3. Customizing the appearance. The properties of the underlying push button is to be visible, but does not print. The background and the border are transparent. The default properties are passed to the push button using a presets command:

```
\newcommand{\overlayPresets}{\H{I}\BG{}\BC{}\S{S}}
```

See the eforms manual for the meaning of these cryptic symbols. You can modify these settings locally, within a group, or globally. In this example, we change the border to red dashed line. We redefine \overlayPresets as follows:

```
1  \renewcommand{\overlayPresets}{\H{I}\BG{}\BC{red}\S{D}}
```

The changes are in line (2), we say \BC{red} (the xcolor package is required here for named colors; otherwise, we would say \BC{1 0 0}), and we've change \S{S} to \S{D}, which gives a dashed (D) border as opposed to a solid (S) border. Now to illustrate this. My name is D. P. Story!; lets increase the viewing window, shall we? My name is D. P. Story!. Keep in mind that we are overlaying a push button; if you want the underlying text to have a color, you need to color it yourself: D. P. Story! This last button has code,

```
\jdRect*[adddestw=10bp,adddesth=10bp]%
    {\textcolor{blue}{D. P. Story}!}
```

As the changes to the preset appearance are inside a group, after this example (environment)) \overlayPresets will revert to its definition that was in effect outside the example. □

5. Special Effects

For the standard set up, where there is a push button that overlays the content along with the viewing window is jumps to, there are two JavaScript "hooks" that can be exploited

- `overlayJmpHook()` is an undefined JavaScript function that is executed after the jump to the viewing window. (It is enclosed in a `try/catch` construct that catches the error thrown.) The document author can define `overlayJmpHook()` to perform some action following the jump. The distribution of fitr comes with one definition, `jmpHookBlink.js`, which blinks the border following the jump.
- `overlayRestoreHook()` is an undefined JavaScript function that is executed following the restored view action. The document author needs to make a custom definition if special effects are desired. The distribution of fitr comes with one definition, `restoreHookBlink.js`, which blinks the border following the restore action.

The preamble of this document says,

```
\usepackage[js=restoreHookBlink,js=jmpHookBlink]{lmacs}
```

The lmacs package is a new package I made available to CTAN, its a simple package that imports files with extensions of `.def`, `.cfg`, and `.js`. We import `restoreHookBlink.js` and `jmpHookBlink.js` using a key-value method, where the key is one of the supported extensions; thus `js=restoreHookBlink` will import the file texttttrestoreHookBlink.js if it exists. By the way, another nice feature of lmacs is that you can prefix an exclamation point (!) to cancel out that import, for example, if we wanted to use `jmpHookBlink` but not `restoreHookBlink` we say

```
\usepackage[!js=restoreHookBlink,js=jmpHookBlink]{lmacs}
```

Example 5.1. Special Effects. Jump to the fitr Package!

The verbatim listing is

```
1 \renewcommand{\overlayPresets}{\H{I}\BG{}\BC{blue}\S{D}}%
2 ...
3 Jump to the \jdRect*[allowFX,adddestw=10bp,adddesth=10bp]%
4     {{\fitrpkg} Package!}
```

We redefined the \overlayPresets command, choosing an initial border of blue. In line (4), I've used allowFX, this key does not normally to appear in the option list, its default value is normally true; however, for this document, the following definition was made in the preamble

```
\renewcommand\allowFXDefault{false}
```

This (re)definition of \allowFXDefault sets the default value of allowFX to false. This was done so the special effects JavaScript functions could be imported (using lmacs) but their effects would not be seen, by default. To see their effect, we have to explicitly put allowFX to true, which is what the single key does. (Or, you can say allowFX=true, but that is five more key presses.) □

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