fitr: Creating FitR Destinations

D. P. Story
Email: dpstory@acrotex.net

processed July 9, 2020

Contents

1 ⟨∗package⟩

Introduction. This package supports dvips and dvipsone utilities, and assumes the PDF creator is **Adobe Distiller**. It also supports the pdflatex (and lualatex) executable through the `pdftex` option; the options `dvipdfm`, `dvipdfmx`, and `xetex` support the corresponding executables.

When using a dvi-ps-pdf work flow (with Distiller as the PDF creator), the package uses the **pdfmark** for the FitR view-type destination:

```
[ /Dest/⟨name⟩/View [ /FitR left bottom right top ]/DEST pdfmark
```

For the `pdftex` or `luatex` option, we use `\pdfdest` a primitive of that utility:

```
\pdfdest name {⟨name⟩} fitr
   width  ⟨wd⟩
   height ⟨ht⟩
   depth  ⟨dp⟩
```

For the `dvipdfm`, `dvipdfmx`, and `xetex` options, the problem of computing the FitR rectangle is a little trickier.

```
\@pdfm@mark{dest (⟨name⟩)
   [ @thispage /FitR @xpos @ypos @urx⟨name⟩ @ury⟨name⟩ ]}}}}
```

where `@urx`⟨*name*⟩ and `@ury`⟨*name*⟩ are the calculated upper right x and y coordinate of the bounding box.

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.

This type of destination allows for the creation of a link or form button to zoom in on a rectangular region.

1 Options and Required Packages

```
\RequirePackage{xkeyval}
\RequirePackage{ifpdf}[2006/02/20]
\RequirePackage{ifxetex}[2006/08/21]
```

(2020/07/03) Added ifluatex package for later testing

```
\RequirePackage{ifluatex}
```

`dvips` `dvipsone` `pdftex` `luatex` `dvipdfm` `dvipdfmx` `xetex`

Driver options. Three options, dvips (the default) and dvipsone, both of which require **Adobe Distiller** as the PDF creator; the pdflatex and luatex allows for the same functionality for the pdflatex executable. We also support `dvipdfm` and its extensions, `dvipdfmx` and `xetex`.

Auto detection of drivers. The drivers pdflatex, lualatex, and xelatex are automatically detected, so there is no need to specify their respective options: `pdftex`, `luatex`, and xetex.

`preview` `viewMagWin`

Preview options. The `preview` option is a carry over from eforms. When selected, all form fields are outlined; useful when setting the location of fields in a dvi previewer. The other option is `viewMagWin` will show the viewing windows surrounding the target. This is the rectangle that will be jumped to. Use this option to adjust the size of the window to your needs. When either of the last two options has an exclamation point prior, that means to turn off the switch. You can conveniently use `viewMagWin` to see the viewing window, then change it to `!viewMagWin` to remove the visible window. Cool. Similarly, you can turn off `preview` using `!preview`.

`!viewMagWin` `!preview`

We create `\ifpreview` and `\ifviewMagWin`, and declare the options.

```
\@ifundefined{ifpreview}{\newif\ifpreview \previewfalse}{}
\@ifundefined{ifviewMagWin}{\newif\ifviewMagWin \viewMagWinfalse}{}
\let\fitr@driver\@empty
```

The `pdftex` option

```
\DeclareOptionX{pdftex}{\gdef\fitr@driver{pdftex}%
  \PassOptionsToPackage{\fitr@driver}{hyperref}%
  \PassOptionsToPackage{\fitr@driver}{eforms}}
\def\fitr@pdftex@driver{pdftex}
```

The `luatex` option

```
\DeclareOptionX{luatex}{\gdef\fitr@driver{luatex}%
  \PassOptionsToPackage{\fitr@driver}{hyperref}%
  \PassOptionsToPackage{\fitr@driver}{eforms}}
\def\fitr@pdftex@driver{luatex}
```

Distiller based drivers.

```
\DeclareOptionX{dvips}{\gdef\fitr@driver{dvips}%
  \PassOptionsToPackage{\fitr@driver}{hyperref}%
  \PassOptionsToPackage{\fitr@driver}{eforms}}
\def\fitr@dvips@driver{dvips}
\DeclareOptionX{dvipsone}{\gdef\fitr@driver{dvipsone}%
  \PassOptionsToPackage{\fitr@driver}{hyperref}%
  \PassOptionsToPackage{\fitr@driver}{eforms}}
\def\fitr@dvipsone@driver{dvipsone}
```

dvipdfm and its variants.

```
\newif \if@fitr@dvipdfm \@fitr@dvipdfmfalse
\DeclareOptionX{dvipdfm}{\gdef\fitr@driver{dvipdfm}%
  \@fitr@dvipdfmtrue
  \PassOptionsToPackage{\fitr@driver}{hyperref}%
  \PassOptionsToPackage{\fitr@driver}{eforms}}
\def\fitr@dvipdfm@driver{dvipdfm}
\DeclareOptionX{dvipdfmx}{\gdef\fitr@driver{dvipdfmx}%
  \@fitr@dvipdfmtrue
  \PassOptionsToPackage{\fitr@driver}{hyperref}%
  \PassOptionsToPackage{\fitr@driver}{eforms}}
\def\fitr@dvipdfmx@driver{dvipdfmx}
\DeclareOptionX{xetex}{\gdef\fitr@driver{xetex}%
  \@fitr@dvipdfmtrue
  \PassOptionsToPackage{\fitr@driver}{hyperref}%
  \PassOptionsToPackage{\fitr@driver}{eforms}}
\def\fitr@xetex@driver{xetex}
```

`preview` `!preview` `viewMagWin` `!viewMagWin`

Various preview options.

```
\DeclareOptionX{preview}{\previewtrue}
\DeclareOptionX{!preview}{\previewfalse}
\DeclareOptionX{viewMagWin}{\viewMagWintrue}
\DeclareOptionX{!viewMagWin}{\viewMagWinfalse}
```

`\previewOn` `\previewOff` `\viewMagWinOn` `\viewMagWinOff`

Some convenience macros to locally turn on or off `preview` and `viewMagWin`.

```
\providecommand{\previewOn}{\previewtrue}
\providecommand{\previewOff}{\previewfalse}
\providecommand{\viewMagWinOn}{\viewMagWintrue}
\providecommand{\viewMagWinOff}{\viewMagWinfalse}
```

`gonative`

When the `gonative` option is specified, field and document JavaScript *are not used*, as a result, no special effects are implemented. This gives a basic document that can be compiled successfully with the dvips -> ps2pdf workflow.

```
\newif\ifFRusedljs \FRusedljstrue
\DeclareOptionX{gonative}{\FRusedljsfalse}
```

`blinkonjmp` `!blinkonjmp` Target region blinks on jump

```
\newif\ifFRblinkonjmp \FRblinkonjmpfalse
\DeclareOptionX{blinkonjmp}{\FRblinkonjmptrue}
\DeclareOptionX{!blinkonjmp}{\FRblinkonjmpfalse}
```

`blinkonrestore` `!blinkonrestore` Target region blinks on restore

```
\newif\ifFRblinkonrestore \FRblinkonrestorefalse
\DeclareOptionX{blinkonrestore}{\FRblinkonrestoretrue}
\DeclareOptionX{!blinkonrestore}{\FRblinkonrestorefalse}
```

`blink` `!blink` A convenience option, `blink` is equivalent to `blinkonjmp` and `blinkonrestore`.

```
\DeclareOptionX{blink}{\FRblinkonjmptrue\FRblinkonrestoretrue}
\DeclareOptionX{!blink}{\FRblinkonjmpfalse\FRblinkonrestorefalse}
```

Before processing options, try for automatic driver detection.

```
\@ifpackageloaded{web}{\ExecuteOptionsX{\eq@driver@name}}{%
  \ifluatex\ExecuteOptionsX{pdftex}\else
  \ifpdf\ExecuteOptionsX{pdftex}\else
  \ifxetex\ExecuteOptionsX{xetex}\else
  \@ifundefined{l@tex@@@@driver}{\ExecuteOptionsX{dvips}}
      {\ExecuteOptionsX{dvipsone}}\fi\fi\fi
}
```

Process the options

```
\ProcessOptionsX
```

Required packages

```
\RequirePackage{xcolor}
```

Minimal packages calc and eforms (which inputs hyperref).

```
%\RequirePackage{hyperref}
\RequirePackage{eforms}[2020/07/05]
\RequirePackage{calc}
```

We also require collectbox, a cool way of collecting an argument which enables the use of verbatim text. Written by Martin Scharrer.

```
\RequirePackage{collectbox}
```

2 The Main Code

Some scratch counters, lengths, boxes.

```
\newcounter{magCnt}
\newbox\fitr@bbox
\newcount\fitr@height
\newcount\fitr@width
\newcount\fitr@depth
\newlength\fitr@length
```

For xelatex we adjust field sizes to conform to the same sizes produced by pdflatex and latex. For lualatex, make a definition so it can be process just like pdflatex.

```
\ifxetex\makeXasPDOff\fi
\ifluatex\protected\def\pdfdest{\pdfextension dest }\fi
```

`\get@fitr@dimen{⟨content⟩}` Tthis command encloses ⟨*content*⟩ in the box register `\fitr@bbox` (an `\hbox`) and transfers dimension info to the count registers `\fitr@height`, `\fitr@width`, and `\fitr@depth`. To typeset ⟨*content*⟩, `\unhbox\fitr@bbox`.

```
\def\get@fitr@dimen#1{%
  \setbox\fitr@bbox=\hbox{#1}%
  \fitr@height=\ht\fitr@bbox
    \xdef\fitr@height@l{\the\ht\fitr@bbox}%
  \fitr@width=\wd\fitr@bbox
    \xdef\fitr@width@l{\the\wd\fitr@bbox}%
  \fitr@depth=\dp\fitr@bbox
    \xdef\fitr@depth@l{\the\dp\fitr@bbox}%
  \setlength\fitr@length{\ht\fitr@bbox+\dp\fitr@bbox}%
  \edef\fitr@@height{\the\fitr@length}%
}
```

dvips. The following commands are some PostScript for the dvips executable.

`\FitRbboxB{⟨amt-wd⟩}{⟨amt-ht⟩}`

```
\def\fitr@urxury@fixup#1#2#3{}
\ifx\fitr@driver\fitr@dvips@driver
\headerps@out{/TeXtoPDF {65536 div DVImag mul} def % sp to pts
/SPtoDvips{TeXtoPDF PDFToDvips} def} % sp to dots
\def\FitRbboxB#1#2{% Uses \fitr@bbox
  currentpoint 2 copy DvipsToPDF \the\fitr@depth\space TeXtoPDF add
  neg vsize add 72 sub #2\space sub exch          % y1
  DvipsToPDF 72 add #1\space sub exch            % x1
  4 2 roll exch DvipsToPDF \the\fitr@width\space
  TeXtoPDF add 72 add #1\space add exch          % x2
  DvipsToPDF \the\fitr@height \space TeXtoPDF sub
  neg vsize add 72 sub #2\space add}             % y2
\else
\ifx\fitr@driver\fitr@dvipsone@driver
```

dvipsone. Definition of `\FitRbbox` for dvipsone.

```
\special{!/TeXtoPDF {65536 div mag 1000 div mul} def
/PDFtoTeX {65536 mul mag 1000 div div} def}
\def\FitRbboxB#1#2{%
   currentpoint 2 copy \the\fitr@depth\space add DvipsToPDF
   neg PageHeight add 72 sub #2\space sub      % y1
   exch DvipsToPDF 72 add #1\space sub exch    % x1
   4 2 roll exch \the\fitr@width\space add
   DvipsToPDF 72 add #1\space add exch         % x2
   \the\fitr@height\space sub DvipsToPDF neg
   PageHeight add 72 sub #2\space add}         % y2
\else\ifpdf
```

pdftex. Definition of `\FitRbbox` for pdftex and luatex. Uses `\fitr@bbox`; `#1` is amount to widen box; `#2` amount to heighten box.

This version of `\FitRbbox` expands the bounding rectangle to the viewing window by increasing the dimensions of width, height and depth. It saves these values in the macros `\fitr@pdftex@view@width|height|depth`.

```
\def\FitRbboxB#1#2{%
  \fitr@length=#1bp
  \fitr@length=2\fitr@length
  \advance\fitr@length\wd\fitr@bbox
  \edef\fitr@pdftex@view@width{\the\fitr@length}%
  \fitr@length=#2bp
  \advance\fitr@length\ht\fitr@bbox
  \edef\fitr@pdftex@view@height{\the\fitr@length}%
  \fitr@length=#2bp
  \advance\fitr@length\dp\fitr@bbox
  \edef\fitr@pdftex@view@depth{\the\fitr@length}%
}
\else\if@fitr@dvipdfm
```

dvipdfm/dvipdfmx/xetex. The primitive `\special{pdf: dest (name)...}` is used along with the primitive variables `@xpos` and `@ypos`. The third argument is ⟨*name*⟩.

```
\def\FitRbboxB#1#2#3{%
  \smash{\raisebox{-\fitr@depth@l-#2bp}%
    {\makebox[0pt][l]{\hspace*{-#1bp}%
```

The values of `@xpos` and `@ypos` correspond to the coordinates of the lower left corner. The values of `@urx#3` and `@ury#3` are calculated in `\fitr@urxury@fixup` as the coordinates of the upper right corner.

```
  \@pdfm@mark{dest (#3) %
    [ @thispage /FitR @xpos @ypos @urx#3\space @ury#3\space ]}}}}}
```

`\fitr@urxury@fixup` is placed just after `\fitr@bbox`.

```
\def\fitr@urxury@fixup#1#2#3{%
  \smash{\raisebox{\fitr@height@l+#2bp}%
  {\makebox[0pt][l]{\hspace*{#1bp}%
```

We raise up to the upper right corner of the content box and define `@urx#3` and `@ury#3` as the current values of `@xpos` and `@ypos`,

```
  \@pdfm@mark{obj @urx#3\space @xpos}%
  \@pdfm@mark{obj @ury#3\space @ypos}}}}%
}
\fi\fi\fi\fi
```

We use a short macro from graphics package. We do not require the graphics command so we redefine it under a different name.

```
\def\fitr@defaultbp#1#2{%
  \afterassignment\fitr@def@bp\dimen@#2bp\relax{#1}{#2}}
\def\fitr@def@bp#1\relax#2#3{%
  \if!#1!%
```

```
   \def#2{#3}%
  \else
    \dimen@.99626\dimen@
    \edef#2{\strip@pt\dimen@}%
  \fi}
```

The major command of this package is `\jdRect`. Its second required argument has several key-value pairs that are recognized. Options for the `\jdRect` command.

`lift` is the amount of lift (performed by `\raisebox`. The value can be a positive or negative length. Positive to translate upward, negative to translate downward. The default is `0pt`.

```
\define@key{fitr}{lift}[0pt]{{\setlength\@tempdima{#1}%
  \xdef\fitr@temp@length{\the\@tempdima}}%
  \edef\fitr@@lift{\fitr@temp@length}}
\def\fitr@@lift{0pt}
```

`shift` is the amount of horizontal shift, positive to the right, negative to the left. The default is `0pt`

```
\define@key{fitr}{shift}[0pt]{{\setlength\@tempdima{#1}%
  \xdef\fitr@temp@length{\the\@tempdima}}%
  \edef\fitr@@shift{\fitr@temp@length}}
\def\fitr@@shift{0pt}
```

`width`
`height` are the dimensions of the bounding box for the button/view rectangle. These dimensions are ignored when the first optional parameter of `\jdRect` is nonempty.

```
\define@key{fitr}{width}[0pt]{{\setlength\@tempdima{#1}%
  \xdef\fitr@temp@length{\the\@tempdima}}%
  \edef\fitr@@width{\fitr@temp@length}}
\def\fitr@@width{0pt}
\define@key{fitr}{height}[0pt]{{\setlength\@tempdima{#1}%
  \xdef\fitr@temp@length{\the\@tempdima}}%
  \edef\fitr@@height{\fitr@temp@length}}
\def\fitr@@height{0pt}
```

`ref` is the reference point. Values are `t` (the default), `c` center, and `b` bottom. When the optional first parameter of `\jdRect` is nonempty, the default is `b`.

```
\define@choicekey+{fitr}{ref}[\val\nr]{t,c,b}[t]%
  {\edef\fitr@@refPt{\val}}{}
\def\fitr@@refPt{t}
```

`adddestw`
`adddesth` add additional length around the (internal) box. This larger rectangle is the view window.

```
\define@key{fitr}{adddestw}[0]{%
  \fitr@defaultbp{\fitr@@adddestw}{#1}}
\def\fitr@@adddestw{0}
\define@key{fitr}{adddesth}[0]{%
  \fitr@defaultbp{\fitr@@adddesth}{#1}}
\def\fitr@@adddesth{0}
```

`button` is a Boolean. If `true` (the default), `\jdRect` creates a push button around the content. When the user pushes the button, the viewer zooms in to the view window. Clicking again restores the previous view. When `button` is `false`, the button is not created, but the viewing window is. You can then jump to the viewing window with a separate link or button.

```
\define@boolkey{fitr}{button}[true]{}
\KV@fitr@buttontrue
```

`link` is an key, which if taken, a link rather than a push button, is created. The `link` key takes one of two values: `jmp`, `restore`.

- When `link=jmp`, a link is created that jumps to the destination determined by `dest`. This value is designed for external links for jumping to a rectangular region Use the key `nodest` to keep `\jdRect` from creating a destination at the link location.
- When `link=restore`, again, a link is created around the content, a restore last view action is associated with the link; clicking on it jumps back to the view at the time you jumped.

```
\define@choicekey+{fitr}{link}[\val\nr]{jmp,restore}[]
    {\edef\fitr@@link{\nr}}{}
\let\fitr@@link\@empty
```

`nodest` is a Boolean, which if `true`, causes `\jdRect` not to generate a destination at the current link (or button).

```
\define@boolkey{fitr}{nodest}[true]{}
\KV@fitr@nodestfalse
```

`dest` When `button=true` (the default), the name of the destination is automatically generated, internally. When `button=false`, you need to specify a destination to reference in a separate link or push button.

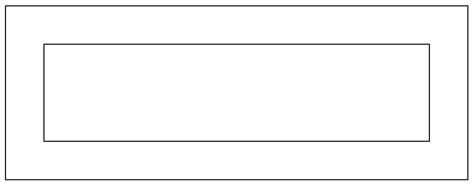
```
\define@key{fitr}{dest}[]{\def\fitr@@dest{#1}}
\let\fitr@@dest\@empty
```

`allowFX` A Boolean switch (of sorts). The `fitr` package 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 are any defined. By saying `allowFX=false`, no special effects are used, even if some are defined.

```
\define@choicekey+{fitr}{allowFX}[\val\nr]{true,false}[true]%
  {\edef\fitr@allowFX{\val}}{}
\newcommand{\allowFXDefault}{true}
\let\fitr@allowFX\@empty
```

`\jdRect` **The main command.** `\jdRect` (optionally) jumps to and/or sets a destination of a FitRectangle.[1] This command is multi-functional. This is the default behavior (`button=true`). The command creates two rectangular regions (which can be viewed using the `preview` and `viewMagWin`).

[1] I had some problems naming this command.



The inner-box represents the bounding rectangle of the push button (or link) that is created when `button=true` (or `link=jmp|restore`). The outer rectangle is the *view window*, the window that is magnified to (the dimensions of this rectangle set by the four FitR parameters. Normally, the inner and outer rectangles are the same, unless the author specifies values of `adddestw` and `adddesth`. The picture above is that can be produced by setting

```
width=2in,height=.5in,adddestw=.2in,adddesth=.2in
```

The inner rectangle may be seen by using the option `preview`, the outer once can be see by using the option `viewMagWin`.

Now if `button=false`, `\jdRect` only sets the destination; it does not create the inner push button. A separate link or push button needs to be created to jump to the view window.

Syntax for `\jdRect` has two forms:

`\jdRect[`⟨*KV-pairs*⟩`]`

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*[`⟨*KV-pairs*⟩`]{`⟨*content*⟩`}`

The second parameter *content* is required when the `*` is present. This version is meant to enclose ⟨*content*⟩ within the button and view window. `width` and `height` are ignored, but `shift` and `lift` are obeyed (though you may `shift` or `lift` the button/view window away from the content).

```
\newcommand{\jdRect}{\begingroup
  \@ifstar{\let\fitr@istar\ef@One\set@rectjd}%
    {\let\fitr@istar\ef@Zero\set@rectjd}}
```

For links, we use the link color of hyperref. The definition of `\ef@colorthislink` is in eforms.

```
\@eqlinktxtcolor{\@linkcolor}
\newcommand{\fitr@bcode}{\ef@colorthislink}%
```

Second stage in the expansion of `\jdRect`.

```
\newcommand{\set@rectjd}[1][]{\def\fitr@jdrect@argi{#1}%
```

We process the key-values before gathering the `\collectbox` so we can insert the link color, if this is a link!

```
  \edef\temp@exp{\noexpand\setkeys{fitr}{#1}}\temp@exp
  \ifx\fitr@@link\@empty\let\fitr@bcode\relax\fi
```

If this is `\jdRect*`, we collect the second argument using `\collectbox`, and move on to `\set@@rectjd`.

```
  \ifx\fitr@istar\ef@One
    \def\fitr@next{\collectbox[\fitr@bcode]{\set@@rectjd}}\else
```

Otherwise, we just move on to `\set@@rectjd`.

```
    \def\fitr@next{\set@@rectjd}\fi\fitr@next}
```

The third and final stage in the expansion of `\jdRect`

```
\def\set@@rectjd{%
  \stepcounter{magCnt}\def\fitr@setBL{0pt}%
  \ifx\fitr@@dest\@empty
    \def\fitr@namedDest{fitrDestn\the\value{magCnt}}\else
    \edef\fitr@namedDest{\fitr@@dest}\fi
  \ifx\fitr@istar\ef@One\def\fitr@@refPt{b}\fi
```

If the user specifies the `link` option, we cancel the `button` option.

```
  \ifx\fitr@@link\@empty\else\KV@fitr@buttonfalse\fi
```

If content is explicitly passed in the optional second parameter we put it in a box and take its measurements. We'll insert the content instead of an empty filled `\parbox`.

```
  \ifx\fitr@istar\ef@One
    \edef\fitr@@width{\the\wd\collectedbox}%
    \edef\fitr@setBL{\the\dp\collectedbox}%
    \setlength{\fitr@length}{\ht\collectedbox+\dp\collectedbox}%
    \edef\fitr@@height{\the\fitr@length}%
  \fi
```

Now, we fit the content into `\fitr@bbox` using `\get@fitr@dimen`.

```
  \ifx\fitr@@link\@empty
    \ifKV@fitr@button\let\fitr@FLB@ction\fitr@OverlayJmpBtn\else
```

`\Bbox` is defined in eforms.

```
    \@ifundefined{ef@Bbox}{\let\fitr@FLB@ction\Bbox}%
      {\let\fitr@FLB@ction\ef@Bbox}\fi
    \else\def\fitr@FLB@ction{\hfill\vfill\@gobbletwo}\fi
```

And overlay the button/link/box

```
  \get@fitr@dimen{%
    \parbox[\fitr@@refPt][\fitr@@height]{\fitr@@width}%
    {\kern0pt\fitr@FLB@ction{\fitr@@width}{\fitr@@height}\kern0pt}}%
```

Now place the button/box according to the user's options.

```
  \makebox[0pt][l]{\hspace*{\fitr@@shift}\smash{%
    \raisebox{\fitr@@lift-\fitr@setBL}{%
    \ifKV@fitr@nodest\else\setFitRDest{\fitr@@adddestw}%
      {\fitr@@adddesth}{\fitr@namedDest}\fi
    \unhbox\fitr@bbox\ifKV@fitr@nodest\else\fitr@urxury@fixup
      {\fitr@@adddestw}{\fitr@@adddesth}{\fitr@namedDest}\fi
  }}}%
  \setlength{\fboxsep}{0pt}%
```

We attempt to represent the bounding box of the view window. We display the bounding view window only if the `nodest` option was not taken, and the `\ifviewMagWin` is true.

```
\ifKV@fitr@nodest\else\ifviewMagWin
  \fitr@length=\fitr@@adddestw bp
  \fitr@length=2\fitr@length
  \addtolength{\fitr@length}{\fitr@@width}%
  \edef\fitr@@width{\the\fitr@length}%
  \fitr@length=\fitr@@adddesth bp
  \fitr@length=2\fitr@length
  \addtolength{\fitr@length}{\fitr@@height}%
  \edef\fitr@@height{\the\fitr@length}%
  \setlength\fitr@length{-\fitr@@adddestw bp}%
  \addtolength\fitr@length{\fitr@@shift}%
  \makebox[0pt][l]{\hspace*{\fitr@length}\smash{%
    \setlength\fitr@length{\fitr@@lift-\fitr@setBL}%
    \if\fitr@@refPt b%
      \addtolength\fitr@length{-\fitr@@adddesth bp}\else
    \if\fitr@@refPt t%
      \addtolength\fitr@length{\fitr@@adddesth bp}%
    \fi\fi
    \raisebox{\fitr@length}%
    {\fbox{\parbox[\fitr@@refPt][\fitr@@height]%
    {\fitr@@width}{\kern0pt\hfill\vfill\kern0pt}}}}%
  }%
\fi\fi
```

If the user passes content through the optional first parameter, we typeset it to the right of the material above (which did not change the position of the current point.

```
\if\fitr@@link\ef@Zero
  \let\fitr@FLB@ction\fitr@OverlayJmpLnk\else
\if\fitr@@link\ef@One
  \let\fitr@FLB@ction\fitr@OverlayRestoreLnk\else
\let\fitr@FLB@ction\relax\let\fitr@bcode\relax\fi\fi
\ifx\fitr@istar\ef@One
  \gdef\fitr@next{\fitr@FLB@ction{\BOXCONTENT}}\else
\let\fitr@next\relax\fi\fitr@next
\endgroup}
```

The code for the **FitR** destination.

```
\def\setFitRDest#1#2#3{%
  \if@fitr@dvipdfm
    \FitRbboxB{#1}{#2}{#3}%
  \else
    \ifpdf
      \FitRbboxB{#1}{#2}%
```

We shift the rectangle `#1bp` to the left without changing the current point. Then use the pdftex primitive `\pdfdest`.

```
      \makebox[0pt][l]{\hspace*{-#1bp}%
        \pdfdest name {#3} fitr
        width  \fitr@pdftex@view@width\space
        height \fitr@pdftex@view@height\space
        depth  \fitr@pdftex@view@depth\space
      }%
```

The **PostScript** code for the **FitR** destination, for the case of `dvips` and `dvipsone`.

```
    \else\literalps@out{%
        [ /Dest/#3/View [ /FitR \FitRbboxB{#1}{#2} ]/DEST pdfmark}%
  \fi\fi
}
```

Form fields. The eforms package allows for what is called presets. We gather some useful options in one group, and pass them all together. `\overlayPresets` sets some defaults for the push button that surrounding the content. The `\restoreOverlayPresets` restores the default definition for `\overlayPresets`.

`\overlayPresets`
`\restoreOverlayPresets`

```
\newcommand{\overlayPresets}{\H{I}\BG{}\BC{}\S{S}}
\newcommand{\restoreOverlayPresets}{%
  \def\overlayPresets{\H{I}\BG{}\BC{}\S{S}}}
```

The `\fitr@OverlayJmpBtn` is a `\pushButton` that is used internally. It calls document JavaScript `pbJmpBtnAction()` for a mouse up action.

```
\def\fitr@OverlayJmpBtn#1#2{%
  \ifx\fitr@allowFX\@empty
    \def\allowFXcode{bAllowFX=\allowFXDefault}\else
    \def\allowFXcode{bAllowFX=\fitr@allowFX}\fi
  \pushButton[\presets{\overlayPresets}\F{-\FPrint}\autoCenter{n}
  \A{\ifFRusedljs\JS{\allowFXcode;\r
    pbJmpBtnAction(event,"\fitr@namedDest","\fitr@namedDest");}\else
    \GoToD(\fitr@namedDest)\fi}]{\fitr@namedDest}{#1}{#2}}

\ifHy@colorlinks\def\pbJmpLnkPresets{}\else
  \def\pbJmpLnkPresets{\Color{\@linkcolor}\W{1}}
\fi
\def\fitr@OverlayJmpLnk#1{\setLink[%
  \presets{\pbJmpLnkPresets}
  \A{\ifFRusedljs\JS{% dps
    pbJmpLnkAction("\fitr@namedDest");}\else
    \GoToD(\fitr@namedDest)\fi}]{#1}}
\def\fitr@OverlayRestoreLnk#1{\setLink[%
  \presets{\pbJmpLnkPresets}
  \A{\ifFRusedljs\JS{% dps
    pbRestoreLnkAction("\fitr@namedDest");}\else
    \Named{GoBack}\fi}]{#1}}
```

3 Document JavaScript

The document JavaScript for fitr.

```
\ifFRusedljs
\begin{insDLJS*}[_fitrLoaded]{fitr}
\begin{newsegment}{JS for the fitr Package}
var _fitrLoaded=true;
var restoreViewState;
var savedRestore=false;
var oSavedRestore=new Object();
var bAllowFX=true;
```

`pbJmpBtnAction(event,fname,dname)` This is the function that the transparent overlay button calls when the user clicks on it.

```
function pbJmpBtnAction(event,fname,dname) {
    if (event.shift) shiftRestoreView(fname,dname);
    else jumpToDest(fname,dname);
}
```

`shiftRestoreView(fname,dname)` The function to call to restore a previously save view.

```
function shiftRestoreView(fname,dname) {
    var bRestore=restoreView(fname,dname);
```

Try built-in store view special effects

```
    if (bAllowFX) try {pbRestoreHook(event,bRestore);}catch(e){}
```

`pbRestoreHookCustom(fname,dname)` A custom function to perform special effects on restore.

```
    if (bAllowFX) try {pbRestoreHookCustom(event,bRestore);}catch(e){}
}
```

`jumpToDest(fname,dname)` When the user clicks on the overlay button, `pbJmpBtnAction()` calls `jumpToDest`. The function stores the current state before jumping.

```
function jumpToDest(fname,dname) {
    if ( typeof oSavedRestore[fname]=="undefined")
        oSavedRestore[fname]=[false,{}];
    else {
```

If `oSavedRestore[fname][0]` is true, that means we have zoomed into the expression, but have not zoomed out again. We'll go ahead and zoom out by calling `shiftRestoreView()`, which does just that.

```
        if (oSavedRestore[fname][0]) {
            shiftRestoreView(fname,dname)
```

When restoring, we return from this call; we don't want to jump to the destination, which is executed later.

```
            return;
        }
    }
    if (!oSavedRestore[fname][0]){
        oSavedRestore[fname][1]=this.viewState;
        oSavedRestore[fname][0]=true;
    }
```

Jump to the named destination.

```
    this.gotoNamedDest(dname);
```

Try the built-in zoom-in special effects.

```
    if (bAllowFX) try { pbJmpHook(event); }catch(e){}
```

`pbJmpHookCustom(event)` is a custom function that is executed as part of the zoom-in action.

```
    if (bAllowFX) try { pbJmpHookCustom(event); }catch(e){}
}
```

`restoreView(fname,dname)` When the user shift-clicks on the overlay button, `pbJmpBtnAction()` calls `restoreView`. This function restores the view before the user magnified the view.

```
function restoreView(fname,dname) {
    if ( typeof oSavedRestore[fname]=="undefined")
        oSavedRestore[fname]=[false,{}];
    if ( oSavedRestore[fname][0] ) {
        this.viewState=oSavedRestore[fname][1];
        oSavedRestore[fname][0]=false;
        return true;
    }
    else return false;
}
```

`pbJmpLnkAction(⟨dname⟩)` is used when the `link=jmp` key-value pair is specified. We jump to the named destination ⟨*dname*⟩. A link does not have a name, so we don't use an array of stored view state as we do for push buttons. We use a single variable for all links.

```
function pbJmpLnkAction(dname) {
    if (!savedRestore){
        restoreViewState=this.viewState;
        savedRestore=true;
    }
    this.gotoNamedDest(dname);
}
```

`pbRestoreLnkAction(⟨dname⟩)` is used when the `link=restore` key-value pair is specified. We restore the previously saved view state.

```
function pbRestoreLnkAction(dname) {
    if (savedRestore){
        this.viewState=restoreViewState;
        savedRestore=false;
    }
}
\end{newsegment}
\ifFRblinkonjmp
```

This script hooks into the DLJS of fitr to blink the field border when the user clicks in it. Length of the blink is 1250 milli-seconds.

```
\begin{newsegment}{Blink border after jump}
```

pbJmpHook(event) blinks the border of an pb push button just after the jmp to focus in on a rectangular region.

```
function pbJmpHook(event) {
    toggleBC.field=event.target;
    toggleBC.startColor=event.target.strokeColor;
    toggleBC.altColor=%
(color.equal(toggleBC.startColor,color.transparent))?%
color.red:color.transparent;
    oSIJ=app.setInterval("toggleBC();",250);
    oTOJ=app.setTimeOut("app.clearInterval(oSIJ); resetBC();",1250);
}
if (typeof toggleBC != "function" ) {
    function toggleBC() {
        var oField=toggleBC.field;
        oField.strokeColor=%
(color.equal(oField.strokeColor,toggleBC.startColor))?%
toggleBC.altColor:toggleBC.startColor;
    }
    function resetBC() {
        toggleBC.field.strokeColor=toggleBC.startColor;
    }
}
\end{newsegment}
\fi % \ifFRblinkonjmp
\ifFRblinkonrestore
```

This script hooks into the DLJS of fitr to blink the field border when the user clicks in it. Length of the blink is 1250 milli-seconds.

```
\begin{newsegment}{Blink border on restore}
```

pbRestoreHook(event,bRestore) Supplies special effects on restore: event is the event object that called this function; bRestore is a Boolean switch, it is true when we are restoring, and false when we are not.

```
function pbRestoreHook(event,bRestore) {
    if (!bRestore) return;
    toggleBC.field=event.target;
    toggleBC.startColor=event.target.strokeColor;
    toggleBC.altColor=%
(color.equal(toggleBC.startColor,color.transparent))?%
color.red:color.transparent;
    oSIR=app.setInterval("toggleBC();",250);
    oTOR=app.setTimeOut("app.clearInterval(oSIR); resetBC();",1250);
}
if (typeof toggleBC != "function" ) {
    function toggleBC() {
        var oField=toggleBC.field;
        oField.strokeColor=%
(color.equal(oField.strokeColor,toggleBC.startColor))?%
toggleBC.altColor:toggleBC.startColor;
    }
    function resetBC() {
        toggleBC.field.strokeColor=toggleBC.startColor;
    }
}
\end{newsegment}
\fi % \ifFRblinkonrestore
\end{insDLJS*}
\fi % \ifFRusedljs
```

⟨/package⟩

4 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

5 Change History

v1.3 (2020/07/03)

General: Add a test for xetex and luatex 4

Added ifluatex package 2

v1.3.1 (2020/07/06)

General: Added `\restoreOverlayPresets` 12

Added `blinkonjmp` and `blinkonjmp` options . . 3

Added `blinkonrestore` and `blinkonrestore` options 4
Added `blink` and `blink` options 4
Added `gonative` option 3
Added commands to locally turn on the `preview` and `viewMagWin` options 3

v1.3.2 (2020/07/09)

General: Bad upload for v1.3.1, corrected problem. 2