Navigator

Paul Isambert
zappathustra@free.fr
01/25/2010

Introduction

Navigator offers access to PDF features such as outlines (bookmarks), links, actions and embedded files; it differs from other existing packages on two main points: first, it doesn't depend on any format and can be used with plain TEX, LATEX, ConTEXt (with some limitations, see here and here), and anywhere else; second, it defines commands to create PDF objects, and can be used as a base to produce raw PDF code across pdfTEX, LuaTEX and XeTEX.

Note that PDF is a description language, and that not all PDF viewers render it completely (not even Adobe Acrobat, actually); thus, some of the features in *Navigator* might seem to produce nothing when the file is read with a reader which doesn't fully support PDF. Fortunately, the most common and useful features, such as links and outlines, are generally supported.

Using *Navigator*

Navigator is loaded in the usual fashion, depending on the format: `\usepackage{navigator}` in LATEX, `\usemodule[navigator]` in ConTEXt, and `\input navigator` anywhere else.

`\finishpdffile` Some of *Navigator*'s operations can take place only at the end of the file. These include producing the outline's hierarchy, sorting embedded files, and writing some information about the document. However, there are exceptions: first in LATEX and ConTEXt, this is done automatically (and for ConTEXt anyway file embedding document settings aren't properly supported; you can do what *Navigator* does with the ConTEXt core, though); second, even if you don't use LATEX or ConTEX t, in XeTEX the outline hierarchy is built at once, so this command isn't needed unless you embed files or want to write some infos to the document's properties. Anyway the command can also be systematically issued harmlessly.

Anchors

Anchors are positions used as the targets of links for navigation within a PDF file (they are called destinations *in PDF parlance). They are to be used in association with the* `\jumplink` *command or with outlines.*

`\anchor` `[<options>]<name>`

This defines *<name>* as an anchor. In vertical mode, the destination is were the command is issued; in horizontal mode, it also depends on the `up` and `left` attributes below. In both cases, however, the value of `fit` might make a difference. The options that follow are also used in the `\outline` command when it defines an implicit anchor.

`up` `<dimension> (Default: \baselineskip.)`

In horizontal mode, anchors are placed on the baseline; it means they send below the intended line (more precisely: with the intended line's baseline at the top of the screen). This attribute makes a vertical correction, so the intended line is shown.

`left` `<dimension> (Default: 0pt.)`

Similar to `up`, but in the horizontal direction (moving leftward).

`fit` `<xyz|fit|fith|fitv|fitb|fitbh|fitbv|fitr> (Default: xyz.)`

All anchors target the page where they appear (fortunately), but when jumping to an anchor how the page is displayed depends on this parameter. The values have the following effect:

- `xyz` The page is displayed so that the anchor is positioned at the upper-left corner of the window. Also, if `zoom` is specified, it is enforced.
- `fit` The page is displayed so that it fits entirely in the window.
- `fith` Same as `fit`, but only horizontally.
- `fitv` Same as `fit`, vertically.
- `fitb` The page is displayed so that its bounding box fits entirely in the window. A page's bounding box is the smallest rectancle enclosing its contents (including headers and footers or marginal material, if any).
- `fitbh` Same as `fith` for the bounding box.
- `fitbv` Same as `fitv` for the bounding box.
- `fitr` Displays the page so that the box where the anchor appears fits entirely in the window. This is disabled with XeTeX, and subtler things might be possible in the future.

zoom <number> (No default.)
: Sets the viewer's zoom when jumping to a destination. The number provided should be 1000 times the intended zoom (like magnification in TEX); hence zoom = 1000 is the normal magnification, i.e. a 100% zoom. The use of this attribute, as well as any value but xyz for the previous one, might be extremely annoying for the reader, because s/he might need to reset his or her zoom after each jump. (Most viewers implement xyz so that it performs its operation as best as possible without zooming; hence, unless the reader already uses a powerful zoom, the way the page is displayed is predictable.)

\anchorname *<name>*
: To avoid possible conflicts, *Navigator* tweaks the names you give it for anchors; if you ever need it, this command returns the real PDF names of the anchor you call *<name>*.

Outlines (aka bookmarks)

Don't hold it against me if I use the words outline *and* bookmarks *alternatively. They are the same things, even though the command is* \outline*. There is no need for a second run to get bookmarks right.*

\outline [*<options>*]*<level>*[*<name>*]*<title>*
: This creates a bookmark with title *<title>*. In LuaTEX, *<title>* is converted to UTF-16 (in octal form, so LuaTEX doesn't complain), so you can use any character of Unicode (although the font used by your reader is very unlikely to display anything besides the basic plane). XeTEX seems to spot the right encoding sometimes. As for pdfTEX, no conversion is performed, so Latin-1 at best may succeed (the string is escaped nonetheless for characters with special meanings in PDF).

  The *<level>* defines how the bookmark fits in the hierarchy. It is a number, and the smaller it is, the higher the bookmark. A bookmark is a child to the nearest preceding bookmark with a smaller *<level>*. The following example illustrates that (hopefully):

```
\outline{1}{A chapter}
  \outline{2}{A section}
    \outline{3}{A subsection}
  \outline{2}{Another section}
```

Now, there is a difference between pdfTeX and LuaTeX on the one hand and XeTeX on the other. First, when using XeTeX, *<level>* should be an integer; no such restriction applies with the other engines. That means that if you suddenly want to insert a bookmark halfway between a chapter and a section in the hierarchy, you don't need to renumber everything: just assign a decimal number to it, for instance `1.5` in the previous example. That is impossible with XeTeX (which will ignore the decimal part, so it is also harmless). Second, with the latter engine, it is also impossible to skip levels, i.e. the following example isn't allowed:

```
\outline{1}{A chapter}
  \outline{3}{A subsection}
```

In that case, XeTeX (i.e. `xdvipdfmx`) will insert an intermediate bookmark, called `<No Title>`, in red and italics (so you definitely can't miss it). With pdfTeX and LuaTeX, since levels are a continuum, skipping a level doesn't make sense and the previous example is perfectly legitimate (however, if a section follows the subsection, its bookmark will appear at the same level as the subsection's). Finally, with all engines, *<level>* can be zero or negative.

Among the *<options>*, those pertaining to a bookmark's appearance are:

open `<true|false> (Default: false.)`
: If set to `true`, the bookmark diplays its immediate children. (You don't need to type '`open = true`'; as with all other boolean attributes, '`open`' suffices.)

bold `<true|false> (Default: false.)`
: If `true`, the bookmark is displayed in a bold font.

italic `<true|false> (Default: false.)`
: If `true`, the bookmark is displayed in an italic font (this is not incompatible with the previous attribute).

color `<red green blue> (Default: 0 0 0, i.e. black.)`
: A triplet of numbers between between 0 and 1 setting the bookmark's color.

outlinecolor <red green blue> (Default: 0 0 0, i.e. black.)
Alias for color. (This name should be used when setting attributes globally in the navigator parameter or elsewhere.)

By default, a bookmark creates an anchor, and clicking it jumps to the position thus defined. If the optional *<name>* is present, then you can refer to that anchor and reuse it, as if you'd issued the \anchor command. Such an anchor, named or not, can take the same options as seen above for the \anchor command, for instance:

```
\outline[zoom = fith]{1}{A chapter}
...
  \outline[left = 2em]{2}[mysection]{A section}
```

Two other options can modify a bookmark's behavior:

anchor <name> (No default.)
As just said, a bookmark creates an anchor. However, this is not true if this attribute is set; in this case, when clicked the bookmark sends to the anchor called *<name>*, which need not be already defined, of course. The attributes pertaining to anchor settings, if any, are ignored, as is the optional name in the \outline command. In other words, the following two bookmarks are equivalent:

```
\outline[anchor = mydest, fit = fitv]{1}[aname]{A title}
\outline[anchor = mydest]{1}{A title}
```

action <name> (No default.)
Like anchor, except the bookmark executes action *<name>* (and doesn't send you anywhere). If both anchor and action are given, the former wins. See the section on actions.

\pdfdef <command><parameter text>{replacement text}
Most of the commands used in TEX will yield nothing good in a bookmark's title. However, you may want to reuse the same text to set that title and, say, a section heading. When redefined with \pdfdef, *<command>* will expand to *<replacement text>*, but only in a bookmark's title. For instance, after:

```
\pdfdef\TeX{TeX}
\pdfdef\emph#1{#1}
```

```
\pdfdef\whatever{\noexpand\whatever}
```

\TeX will produce "TeX" in a bookmark, \emph will simply return its argument, and \whatever will represent itself. Note that unexpandable commands and \protected commands don't require any special treatment to represent themselves.

Links

Links (and annotations more generally) are clickable zones of a PDF document, which trigger actions. Navigator *offers a few types of links and actions, and also allows you to create your own with raw PDF objects. The <options> that appear in the commands below are described at the end of this section.*

\jumplink [*<options>*]*<name>*<*text*>

This prints *<text>* and defines it as a clickable zone that sends to the anchor called *<name>* (defined either with \anchor or \outline, although the anchor need not be already defined for the link to work, obviously). For instance, clicking \anchor or \outline in the previous parenthesis sends you to the page where those commands are defined.

\urllink [*<options>*]*<url>*<*text*>

This is the same thing as \jumplink, except the link sends to an internet resource (any URI works, actually). For instance :

```
\urllink[border = 1, color = 1 0 0, dash = 3 2]
        {http://www.tug.org}{Go to TUG!}
```

produces : Go to TUG! If the uri or base attributes in the navigator parameter are set, then *<url>* can be relative.

\javascriptlink [*<options>*]*<JavaScript code>*<*text*>

A link which executes some JavaScript code (if the viewer can do that). For instance :

```
\javascriptlink[highlight = push]
               {app.alert("Hello!")}{Say hello!}
```

produces : Say hello!

`\actionlink` `[<options>]<name><text>`

This executes the action called *<name>*. A named action is an action that is defined elsewhere in the document (not necessarily before) and given a name to refer to it. The next section introduces two simple commands to create URL and JavaScript actions; however, an action is just a PDF object; hence, *<name>* can be any valid PDF object (of the right type, obviously), and you can define such objects with the commands introduced in the last section of this document. *Navigator* already defines four actions, namely: firstpage, lastpage, prevpage and nextpage. The meaning should be obvious, but if you have any doubt, just click!

`\rawactionlink` `[<options>]<PDF code><text>`

This is the same as `\actionlink`, except the action is defined not in an external action but in *<PDF code>*, which should be an action dictionary (without the enclosing << and >>, exactly as with `\pdfdictobject`).

In the commands above, the *<options>* are used to set the appearance of the links. Here are the ones you can set:

`border` `<number>` `(Default: 0.)`

The width (in PostScript points) of the border drawn around the link. Default 0 means no border. (In pdfTeX and LuaTeX, you can use the `\pdflinkmargin` primitive to set the distance between the link's border and its content, even if the border isn't drawn – it makes the clickable zone larger.)

`color` `<red green blue>` `(Default: 0 0 0, i.e. black.)`

The color of the link's border, if any.

`linkcolor` `<red green blue>` `(Default: 0 0 0, i.e. black.)`

Alias for `color`. (This name should be used when setting attributes globally in the `navigator` parameter or elsewhere.)

`dash` `<numbers>` `(No default.)`

The dash pattern of the link's border, can you believe it? The *<numbers>* should be `a1 b1 a2 b2...` where a's specify visible parts of the border in PostScript points and b's specify hidden parts. Then the pattern repeats itself. More or less. (Set `dash` to `1 0` if you want to override a default dash pattern.)

highlight <none|invert|outline|push> (Default: invert.)
This defines how the link flashes when clicked: none does nothing, invert inverts its colors, outline inverts its border's colors if any, and push gives the amazing illusion that the link is pushed below the surface of the page (says the author of the *PDF reference*, except s/he didn't mention any amazing illusion).

pre <code> (No default.)
TEX code to be appended before *<text>*.

post <code> (No default.)
TEX code to be appended after *<text>*. This and pre allows you to achieve coherent formatting of links. For instance:

```
\def\weblink{\urllink[pre = \bgroup\bf, post = \egroup]}
Go to \weblink{http://www.tug.org}{TUG} and
upload to \weblink{http://ctan.org}{CTAN}.
```

produces: Go to **TUG** and upload to **CTAN**.

raw <PDF code> (No default.)
Raw PDF code to be inserted in the link's dictionary.

\annotation [*<options>*]*<PDF code><text>*
This creates an annotation tied to *<text>*. *<PDF code>* goes into the annotation's dictionary; note that the annotation thus created doesn't even have a Subtype entry (a link, as described in this section, is one of the many subtypes of annotation). Depending on the *<options>*, the Border entry, the C entry (for color) and the H (for highlight) might be specified.

There is one more type of action link, \openfilelink; it takes the same options as the commands described here, but it is introduced in the section on embedded files, because it is related to them.

Actions

The commands in this section allow you to create actions and refer to it somewhere else in your document. That is a convenient approach for actions that are used repeatedly. Note that the PDF file itself is also improved, because in it too actions thus defined appear only once.

You might think that not much types of actions can be designed: only internet addresses and JavaScript. But those are only predefined patterns to be used even when one doesn't know anything about PDF. Other types of actions require that you write real PDF code, more precisely so-called action dictionaries; these are but dictionary objects, and Navigator *allows you to create them with* \pdfdictobject*. Then the name you give to that object is a valid action name and can be used with* \actionlink*.*

In other words, there is no \whateveraction *command, because that's what* \pdfdictobject *does.*

\urlaction *<name><url>*

Defines *<name>* as an action which sends to the address *<url>* on the internet. (Actually, any URI can be used, not only URLs.) As with \urllink, if the uri or base attributes in the navigator parameter are set, then *<url>* can be relative.

\javascriptaction *<name><JavaScript code>*

Defines *<name>* as an action which executes *<JavaScript code>* (if the viewer can do that).

Thus the following are equivalent to the previous two examples with \urllink and \javascriptlink:

```
\urlaction {tug}{http://www.tug.org}
\actionlink{tug}{Go to TUG!}

\javascriptaction{hello}{app.alert("Hello!")}
\actionlink{hello}{Say hello!}
```

Embedded files

Navigator *provides a simple command to embed files in a PDF document. Embedded file can also be opened with an action link, but only if they are PDF files. However, no proper support is offered for file embedding in ConTEXt.*

\embeddedfile [*<description>*]*<object name>*[*<alternate filename>*]*<file>*

This embeds *<file>* in the current document; the name displayed by the viewer will be *<file>* or *<alternate filename>*, if used (do not forget the file's extension in that alternate name). The use of an alternate name is useful if *<file>* contains a path. The viewer might also display an optional *<description>*. (The command name \embeddedfile, cumbersome when compared

to the more obvious \embedfile, is meant to avoid a possible conflict with Heiko Oberdiek's embedfile.)

The command creates a PDF object called *<object name>*; its Type is Filespec, and it points to another object of type EmbeddedFile (containing the actual file). That object isn't used anywhere in *Navigator* for the moment, but it is available in case you need it, for instance when writing raw PDF.

\openfilelink [*<options>*]*<file>*[*<page>*]*<text>*

This is an action link, so see the description of the *<options>* above. When clicked, it opens *<file>* on page *<page>* (first page if *<page>* is omitted). That file should be embedded elsewhere in the document with \embeddedfile, and it should be a PDF file; *<file>* should match the *<file>* argument in \embeddedfile, or *<alternate filename>* if given.

Settings

Many of the commands introduced in the previous sections can take options. If an option is not given, its default value is used instead; you may want to set those default values, and to do so you have to change the attributes of the navigator *parameter:*

```
\setparameter navigator:
  <attribute> = <value>
  <attribute> = <value>
...
\par
```

This is the YaX syntax, it is a little bit special, and if you don't want to learn it the traditional comma-separated list is also possible:

```
\setparameterlist{navigator}
                 {<attribute> = <value>,
                  <attribute> = <value>...}
```

(If a *<value>* is true it can be omitted.) Learning YaX might still be a good idea, though, since then you can use the meta attribute, which wasn't shown when the options for the commands above were explained, because it can be used anywhere. For instance:

```
\setparameterlist{mylink1}
                 {meta  = navigator,
```

```
                    color = 1 0 0}
\setparameterlist{mylink2}
                  {meta  = navigator,
                   color = 0 1 0}
...
\actionlink[meta = mylink1]{...}{...}
\actionlink[meta = mylink2]{...}{...}
```

There you define two types of links, and you can change their settings at once by changing the parameters `mylink1` and `mylink2` instead of the links themselves (which might be numerous). Note how `mylink1` and `mylink2` define `navigator` as the meta-parameter. That is not necessary, but the action links wouldn't inherit your global default settings otherwise (unless the meta-parameter of `mylink1` and `mylink2` itself had `navigator` as its meta-parameter, or a meta-parameter which... got it?). The `navigator` parameter itself can have a meta-attribute too.

At last, here are the attributes that can be set (some make sense in individual commands only, and not in the `navigator` parameter, but you can still set them):

`up`, `left`, `fit`, `zoom`: see the section on anchors (note that they also affect outlines when outlines create implicit anchors).

`outlinecolor`, `open`, `bold`, `italic`, `action`, `anchor`: those are options for outlines. (In a given outline, one can use the `color` attribute instead of `outlinecolor`; in the `navigator` parameter, however, one should use only the latter.)

`border`, `linkcolor`, `highlight`, `dash`, `raw`, `pre`, `post`: those affect action links. (The remark on `outlinecolor` above holds for `linkcolor` here.)

And here are attributes that aren't associated with commands, but instead specify options and pieces of information for the entire document. Note that they do not work in ConTEXt, but that can be done easily without *Navigator*.

`author` `<string> (No default.)`
The author of the document. This shows up in the document's properties. The string is dealt with as described for an `\outline`'s title.

`title` <string> (No default.)
: Same as `author`, with the document's title.

`keywords` <string> (No default.)
: Same thing again, with keywords.

`subject` <string> (No default.)
: One more time for the world, this time with the subject addressed by your document.

`date` <date> (Default: current date.)
: The creation date; the date should be formatted as explained in the *PDF Reference*, so have fun. Anyway that is set automatically by TEX (and can't be set with XeTEX).

`moddate` <date> (Default: current date.)
: Same as `date`, for the modification date.

`creator` <string> (Default: TeX.)
: The software that produced the document's original format.

`producer` <string> (Default: pdfTEX, LuaTEX or XeTEX.)
: The software that turned the original document into PDF (cannot be set with XeTEX).

`rawinfo` <PDF code> (No default.)
: Entries you want to add by yourself to the document's `Info` dictionary.

`layout` <onepage | onecolumn | twopage | twocolumn | twopage* | twocolumn*> (No default.)
: This sets how the document should be displayed when opened:

  `onepage` The window displays only one page, with no continuous transition to the next one.

  `onecolumn` The window displays only one page, with a continuous transition to the next.

  `twopage` The window displays two pages, odd-numbered pages on the right, without a continuous transition.

  `twocolumn` Same as `twopage` with a continuous transition.

  `twopage*` Same as `twopage` with odd-numbered pages on the left.

  `twocolumn*` Same as `twopage*` with a continuous transition.

mode <outlines | bookmarks | thumbnails | thumbs | attachments | files | oc>
(No default.)
This sets what panel the reader should display when the document is opened:

outlines	The reader shows the document's outlines.
bookmarks	Same as outlines.
thumbnails	Thumbnail images are shown.
thumbs	Same as thumbnails.
attachments	Show the attached files.
files	Same as attachments.
oc	The Optional Content Group panel is displayed; quite useless if you don't have any OCG's (*Navigator* might offer support one day).

uri <address> (No default.)
The base address that will be used with relative references. See \urllink and \urlaction.

base <address> (No default.)
Alias for uri.

openaction <name> (No default.)
The *<name>* of the action to be performed when the document is opened.

rawcatalog <PDF code> (No default.)
Raw PDF code to be added to the document's catalog.

PDF objects

Here are commands that let you create and use PDF objects (and which Navigator *actually uses internally). Of course they are useless unless you know how PDF works (if you do, then you might note that what I call here* objects *actually are* indirect objects*).*

Named actions used with \actionlink being objects, they behave as any other object with the commands below; for one thing, they respond positively to \ifpdfobject, and you can't create an object with the same name.

\pdfobject *<name><code>*
Creates a raw PDF object with name *<name>*. There shouldn't already exist an object with the same name (unless it is just reserved).

\pdfdictobject *<name><code>*
Same as \pdfobject, except a dictionary object is created. This command is totally equivalent to the previous one with *<code>* enclosed between << and >>. With this you can write custom named actions, i.e. *<name>* can be used with \actionlink.

\pdfstreamobject [*<raw code>*]*<name><stream>*
Creates a PDF stream object, with optional *<raw code>* added to the dictionary.

\pdffileobject [*<raw code>*]*<name><file>*
Same as \pdfstreamobject, except the stream is the contents of *<file>*.

\pdfreserveobject *<name>*
Reserves an object with name *<name>*. That is totally useless (but harmless too) with XeTEX. In pdfTEX and LuaTEX, it is needed to refer to an object that isn't created yet.

\pdfensureobject *<name>*
If *<name>* isn't an object (reserved or created), this command reserves it. Otherwise it does nothing.

\pdfobjectnumber *<name>*
Returns the number of object *<name>* (which must be reserved at least). In XeTEX, it actually returns nothing, because you can't use object numbers there, but fortunately in pdfTEX and LuaTEX it isn't very useful either; the next command is much more interesting. (If *<name>* doesn't exist, this command returns 0; an object can't have number 0 in PDF, but that makes more sense than returning an error message, which would never make it to the terminal and spoil the PDF file instead.)

\pdfrefobject *<name>*
Makes an indirect reference to object *<name>*. For instance:

```
\urlaction{tug}{http://www.tug.org}
```

and then somewhere while writing PDF code:

```
/A \pdfrefobject{myobj}
```

(As with \pdfobjectnumber, if *<name>* doesn't exist, i.e. it hasn't

been reserved or created, this command returns `0 0 R`, an impossible object that is better than an error message. With XeTeX, though, the reference is made anyway.)

`\pdfobjectstatus` *<name>*
Returns 0 if *<name>* isn't an object, 1 if it is a reserved object and 2 if the object has been created.

`\ifpdfobject` *<name><true><false>*
Executes *<true>* if *<name>* is an object (reserved or created) and *<false>* otherwise.

`\pdfstring` *<string>*
This transforms *<string>* into a valid PDF string: in XeTeX this actually does nothing, whereas in pdfTeX the string is escaped, and in LuaTeX it is converted to UTF-16 in octal form. Note that this operation is already performed in a bookmark's title, in the description of an embedded file, in `\javascriptaction` and in various attributes of the `navigator` parameter, so you shouldn't use it there; it might be otherwise useful when writing raw PDF code.

Typeset with LuaTeX v.0.66
in Lucida and Lucida Console
(Charles Bigelow and Kris Holmes)