The PracTEX Journal, 2007, No. 2
Article revision 2007/05/16

PDFpages for Editors and Publishers

Yuri Robbers

Email yuri.robbers@gmail.com

Abstract There are many ways in which the PDFpages package (Matthias, 2006) can be helpful to editors. An obvious one is collating several papers into one. This paper will describe a few of the many ways in which pdfpages can make life easy for the editor and publisher.

1 Introduction

Editors often need to manipulate existing PDF documents in various ways that would be difficult, time consuming, or in extreme cases downright impossible in basic LaTeX2$_\varepsilon$. One could think of concatenating various papers into one large PDF (possibly even with hyperlinks from a table of contents to each included paper), shrinking or enlarging pages, rearranging the pages of a document (for example into signatures, for printing as a booklet), creating a new document that is a subset of the original document or including pages of a document as illustrations in another document (perhaps the front covers of books in a catalog). There are, of course, many more possible applications.

When faced with such tasks, the life of an editor suddenly becomes a lot easier with the PDFpages package (Matthias, 2006) at their disposal.

In this paper I will show a few of the many possible applications of PDFpages.

2 Concatenating papers

It happens a lot when editing journals, such as the PracTEX Journal. One has a whole bunch of papers, and they need to be combined in a single "whole issue" PDF. Since the PracTEX Journal consists of materials written in TEX, LaTeX2$_\varepsilon$, ConTEXt and HTML this is not necessarily a simple task. We cannot just create one big TEX source file and compile it. Even when using just LaTeX2$_\varepsilon$ it is possible to run

into trouble when one paper contains a package or style file that is incompatible with a package or style file in another paper.

In order to avoid these problems it is often easiest to create a PDF of each paper, and concatenate those. The traditional way of doing this uses Adobe® Acrobat Professional. This program is still one of the best (if not the best) at converting HTML into PDF, and also allows concatenating PDFs into one large PDF by mouseclick.

Apart from being free software the PDFpages package, however, has a very important advantage to offer, even in this simple case: programmability. The PDFpages package is not able to turn HTML into PDF, so that task will have to be performed using a different program, but the next step — concatenation — profits from programmability. Suppose one has forgotten to include a certain paper. Or one of the proofreaders (or worse: readers) has discovered a glaring error in one of the papers. With the mouse click approach to concatenation one will have to do the browsing of directories and clicking of files all over again. With another opportunity for mistakes. With the PDFpages approach, one simple types

```
pdflatex wholeissue.tex
```

and the whole issue is regenerated. Quick and easy, with no chance of mistakes. The only thing one needs to do is create the `wholeissue.tex` file once, and then concatenation is a simple matter of running the command above.

An example of such a `wholeissue.tex` file is given in figure 1. As one can see, apart from the usual preamble, the document consists of a single command `\includepdfmerge`. This command has an argument with a whole list of PDF filenames, followed by a comma, a minus sign, and another comma. The minus sign is the notation the PDFpages package uses to signify "all pages". It is possible to give a single number here in order to include just one page, or a page range, or even a complex list of pages, such as `1,3,4,8-12,16,15,14,19-22,35,1,1,1`. As one can see, it is even possible to include pages in a different order, or to include the same page multiple times.

Please note that the last paper that is being included does *not* have a comma after the minus sign: the comma signifies that either a further reference to a page, or another file to included will follow.

Since this example is part of the concatenation file for the 2007-1 issue of the PracTEX Journal, it contains two PDF files for most papers, since those papers come as a PDF file with an additional HTML file containing an abstract of

```
\documentclass{amsart}
\usepackage{pdfpages}
\begin{document}
\includepdfmerge[fitpaper=true]{%
contents.pdf,-,%
editorial.pdf,-,%
news.pdf,-,%
readers.pdf,-,%
beccari-html.pdf,-,%
beccari.pdf,-,%
gregorio-html.pdf,-,%
gregorio.pdf,-,%
robbers-html.pdf,-,%
robbers.pdf,-,%
madsen-html.pdf,-,%
madsen.pdf,-,%
cho-html.pdf,-,%
cho.pdf,-,%
blaga-html.pdf,-,%
blaga.pdf,-,%
baechle-html.pdf,-,%
baechle.pdf,-,%
neveln-html.pdf,-,%
neveln.pdf,-,%
asknelly.pdf,-,%
distract.pdf,-%
}
\end{document}
```

Figure 1: An example `wholeissue.tex` file that will concatenate a series of PDF files into one whole issue PDF for an online journal.

the paper and a short biography of the author. Both files are included in the `wholeissue.pdf` file.

Should one discover that an article has been missed in the whole issue PDF, one can easily add that file to the `wholeissue.tex` and regenerate the whole issue PDF. Similarly, it is easy to change the order of the papers in the whole issue PDF, or to regenerate the whole issue PDF when the PDF file containing one of the papers has been changed.

3 Producing screen and print versions of eBooks

There is a small independent publisher[1] in the Netherlands that produces eBooks, which I typeset for them in LaTeX2$_\varepsilon$, using Peter Wilson's memoir document class (Wilson, 2004).

As an extra service to customers each eBook is presented in a so-called *screen version* as well as a *print version*. The screen version contains the front cover, the copyright page, the story itself and the back cover. It is intended to be read on a computer screen or a dedicated eBook reader such as the iLiad[2] or Sony Reader[3].

The print version contains the front cover, title page, copyright page, story and back page. Blank pages have been added wherever they are appropriate, as well as at the end of the story, if necessary, to make sure the total number of pages is a multiple of four. The pages have then been ordered into so-called signatures: an ordering of the pages different form the usual numerical one with two pages being printed on each side of a sheet of paper. This PDF file can be printed double sided on a printer, folded in half and — if desired — stapled through the back and/or cut to size. This way one has a small booklet which can easily be taken along in a pocket.

Both the screen version and the print version are being generated by means of the appropriate PDFpages file. Given a PDF file containing the title page, copyright page and story (generated by pdfLaTeX based on my memoir file), a PDF file containing the front cover, a PDF file containing the back cover, and a PDF file containing a blank page, I can generate the screen version with one command, and the print version with two.

1. Saga-Whyte Press, `http://www.sagawhytepress.eu`
2. `http://www.irextechnologies.com` and `http://www.sagawhytepress.eu`
3. `http://www.learningcenter.sony.us/assets/itpd/reader/`

3.1 Screen version

Suppose there is a directory for each eBook — "*Sheena and the Gun*" (Skjold, 2007b) shall be our example story — containing all the relevant files. Among the files in this directory is the PDF file of the story itself called `sheena.pdf`. The story file starts with a title page, then a copyright page, and then the story itself. In this directory there are also PDF files for the cover, aptly named `frontcover.pdf` and `backcover.pdf`.

In order to create the screen version of this eBook, all we need to do is run this PDFpages file through pdfLaTeX:

```
\documentclass{amsart}
\usepackage{pdfpages}
\begin{document}
\includepdfmerge[fitpaper=true]{%
frontcover.pdf,%
sheena.pdf,2-,%
backcover.pdf%
}
\end{document}
```

Since we already have the front cover, a title page is not truly necessary in the screen version of the eBook. That is why the story file itself is included from page 2 onwards. Also note that for PDF files that contain only one page, it is possible (though not at all necessary) to leave out the page range designation.

3.2 Print version

Creating the print version of the same eBook unfortunately requires two runs of pdfpages. The reason is that if one combines the concatenation of the files and the re-ordering of the pages into a signature in one run of PDFpages, one gets extraneous white space in various places. There might be a more elegant solution to this than mine, but I have not been able to find it.

My solution entails two runs of pdfTeX. The first one takes care of concatenation:

```
\documentclass{amsart}
\usepackage{pdfpages}
```

```
\begin{document}
\includepdfmerge[fitpaper=true]{%
frontcover.pdf,%
blankpage,-,%
sheena.pdf,-,%
blankpage,1,1,1,%
backcover.pdf%
}
\end{document}
```

where I include the blank page between the story and the back cover three times by including its first (and only) page three times. The first two are to make sure that the total number of pages is divisible by four, and the third one is the inside back cover (just as the very first blankpage between the front cover and the story is the inside front cover).

Then I run the resulting PDF file through PDFpages again, in order to take care of the re-ordering:

```
\documentclass{amsart}
\usepackage{pdfpages}
\usepackage[paperheight=250mm,paperwidth=168mm]{geometry}
\begin{document}
\includepdf[pages=-,landscape,signature=28]{sheena-concatenated.pdf}
\end{document}
```

There are a few changes to be seen here. First of all in the preamble. I use the geometry package in order to set the paperheight to equal twice the pagewidth and the paperwidth to equal the pageheight. This way I can turn the pages and fit two of them together onto one sheet.

Next I use `\includepdf` rather than `\includepdfmerge` because I am not merging files, just using the one file. When using `\includepdf` one includes the page specification in the optional argument. The other optional argument, landscape, switches width and height of the paper. For reasons unknown to me one cannot just refrain from exchanging paperwidth and -height and from using the landscape option, because then the final result will have extraneous white again. The option `signature=28` tells PDFpages that one wants a bunch of 28 pages to be generated in such a way that the resulting printout can be folded in half and have

a booklet with all the pages in the right order.

One could of course specify a larger number than the number of pages present. In that case PDFpageswould add blank pages at the end of the document. In this example that could not be used, since the back cover needed to be included as the very last page.

Specifying a smaller number than the number of pages is a very useful function too in printing. When looking at a printed book, a novel for example, one sees that the book actually consists of a number of small booklets stacked on top of each other. Each booklet has 32, or sometimes 16 pages. This effect is achieved by setting `signature=32` or `signature=16` respectively. The printer would churn out stacks of booklets of 32 or 16 pages each, that can be sown, then stacked on top of each other and finally glued together into the cover, which must have an actual spine in this case rather than just being the front and back cover printed together on one sheet.

4 Previews of books

Something else that can useful for a publisher is the creation of a preview. Creating a preview of a short story works much like creating the screen or print version of a booklet as shown above. The main difference is that one includes only part of the main book file. For a screen preview of the short story used above, one would include the front cover, the copyright page, the first two or three pages of the story, perhaps a page with information about the author and publisher, and where to obtain the full book, and then the back cover.

For a print preview one would do the same, but include the correct number of blank pages wherever appropriate, and then run a second pass in order to create a signature PDF file.

Take, for example, the novel "*Scent of Summer Magnolia*" (Skjold, 2007a). This is a novel consisting of 560 pages, which includes the frontmatter, mainmatter and backmatter. In addition, there is the front cover, back cover and spine. For the preview we will not be using the spine. We want to include both the covers, the front matter (title page, copyright page, table of contents, etc.), the first two chapters of the novel itself, and the back matter ("*About the author*"). The concatenation file would look as follows:

```
\documentclass{amsart}
\usepackage{pdfpages}
\begin{document}
\includepdfmerge[fitpaper=true]{%
frontcover.pdf,%
blankpage.pdf,%
magnolia.pdf,1-18,559,560,%
blankpage.pdf,%
backcover.pdf%
}
\end{document}
```

Note that here we do not need to have any additional blank pages, since our selection of pages already comes to a total number divisible by four.

The creation of the signature file requires just changing the number of pages to be included in the signature (24), the papersize and the filename in the signature file given at the end of subsection 3.2:

```
\documentclass{amsart}
\usepackage{pdfpages}
\usepackage[paperheight=12in,paperwidth=9in]{geometry}
\begin{document}
\includepdf[pages=-,landscape,signature=24]{magnolia-concatenated.pdf}
\end{document}
```

And this way one has a booklet that can be printed, folded, stapled and given away to potential customers in order to entice them into buying the book.

5 Including (parts of) documents as illustrations

The final useful example I will give here is the inclusion of existing documents or parts of existing documents as illustrations in a new document. This can be useful when discussing, for example, an existing text or figure from a publication available in PDF format, or in order to display covers of books (in a catalog perhaps) or journals (in an index for a complete volume of that journal perhaps.

In this section I will show how to do such an inclusion using PDFpages. As an example I will use part of the PDFpages documentation (Matthias, 2006). Of

course I will not be including all pages, but I will include four, in order to show one page of thumbnails.

This code

```
\includepdf[nup=2x2,pages={1-4},frame=true,%
noautoscale,scale=.35]{pdfpages.pdf}
```

produces the last page of this document. Please note that PDFpages package creates *pages* rather than graphics objects, so they cannot be put inside floats, or otherwise be made just part of a page rather than an entire page.

6 Conclusion

The PDFpages package has many useful applications, especially for editors. This paper shows a few of them. Do read the PDFpages documentation (Matthias, 2006) for further information, and experiment, for it is practice that makes perfect.

7 Acknowledgments

I am grateful to Dave Walden for his comments on a draft of this paper.

References

Matthias, A. (2006). The pdfpages package. CTAN://macros/latex/contrib/pdfpages.

Skjold, A. (2007a). *Scent of Summer Magnolia*. Leiden, the Netherlands: Saga-Whyte Press.

Skjold, A. (2007b). *Sheena and the Gun*. Leiden, the Netherlands: Saga-Whyte Press.

Wilson, P. (2004). The memoir class for configurable typesetting. CTAN://macros/latex/contrib/memoir/.

The pdfpages Package*

Andreas MATTHIAS
amat@kabsi.at

2006/08/12

Abstract

This package simplifies the insertion of external multi-page PDF or PS documents. It supports pdfLaTeX and VTeX.

Contents

1 Introduction

When creating PDF documents, it is sometimes useful to insert pages of other, external PDF documents. This can be done with the `\includegraphics` command from the `graphics` package. But a simple `\includegraphics{doc.pdf}` normally produces `Overfull \hbox` and `Overfull \vbox` warnings, because the size of the inserted pages does not match the print space.

The `pdfpages` package makes it easy to insert pages of external PDF documents without worrying about the print space. Here are some features of the `pdfpages` package: Several logical pages can be arranged onto each sheet of paper and the layout can be changed individually. A lot of hypertext operations are supported, like links to the inserted pages, links to the original PDF document, threads, etc. When working with VTeX the same is possible with PostScript documents, too. Note that PostScript documents are only supported by VTeX and *not* by pdfLaTeX.

*This file has version number v0.4a, last revised 2006/08/12.

When producing DVI output `pdfpages` cannot insert pages of a PDF documents. But instead of interrupting execution `pdfpages` will insert empty pages. This feature is important when using packages like `pst-pdf`, which need to produce DVI output at the first run.

2 Usage

2.1 Package Options

`\usepackage[`⟨*options*⟩`]{pdfpages}`

⟨*option*⟩ – **`final`:** Inserts pages. This is the default.

`draft`: Does not insert pages, but prints a box and the filename instead.

`enable-survey`: Activates survey functionalities. (*experimental, subject to change*)

2.2 Commands

`\includepdf` Inserts pages of an external PDF document.

`\includepdf[`⟨*key=val*⟩`]{`⟨*filename*⟩`}`

⟨*key=val*⟩ – A comma separated list of options using the ⟨*key*⟩=⟨*value*⟩ syntax.

⟨*filename*⟩ – Filename of the PDF document. (The filename *must not* contain any blanks!)

The following list describes all possible options of `\includepdf`. All options are using the ⟨*key=value*⟩ syntax.

- Main options:

pages Selects pages to insert. The argument is a comma separated list, containing page numbers (`pages={3,5,6,8}`), ranges of page numbers (`pages={4-9}`) or any combination. To insert empty pages use `{}`.
E.g.: `pages={3,{},8-11,15}` will insert page 3, an empty page, and pages 8, 9, 10, 11, and 15.
Page ranges are specified by the following syntax: ⟨*m*⟩-⟨*n*⟩. This selects all pages from ⟨*m*⟩ to ⟨*n*⟩. Omitting ⟨*m*⟩ defaults to the first page; omitting ⟨*n*⟩ defaults to the last page of the document. Another way to select the last page of the document, is to use the keyword `last`. (This is only permitted in a page range.)
E.g.: `pages=-` will insert *all* pages of the document, and `pages=last-1` will insert all pages in reverse order.
(Default: `pages=1`)

nup Puts multiple logical pages onto each sheet of paper. The syntax of this option is: `nup=`⟨*xnup*⟩`x`⟨*ynup*⟩. Where ⟨*xnup*⟩ and ⟨*ynup*⟩ specify the number of logical pages in horizontal and vertical direction, which are arranged on each sheet of paper. (Default: `nup=1x1`)

landscape Specifies the format of the sheet of paper, which is rotated by 90 degrees. This does *not* affect the logical pages, which will *not* be rotated by the 'landscape' option. To rotate the logical pages use the 'angle' option (e.g. 'angle=90'). Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `landscape=false`)

- Layout options:

delta Puts some horizontal and vertical space between the logical pages. The argument should be two dimensions, separated by space. See Chapter 2.3 and Figure 1. (Default: `delta=0 0`).

offset Displaces the origin of the inserted pages. The argument should be two dimensions, separated by space. In 'oneside' documents positive values shift the pages to the *right* and to the *top* margin, respectively, whereas in 'twoside' documents positive values shift the pages to the *outer* and to the *top* margin, respectively. See Chapter 2.3 and Figure 1. (Default: `offset=0 0`)

frame Puts a frame around each logical page. The frame is made of lines of thickness `\fboxrule`. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `frame=false`)

column Pdfpages normally uses 'row-major' layout, where successive pages are placed in rows along the paper. The `column` option changes the output into a 'column-major' layout, where successive pages are arranged in columns down the paper. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `column=false`)

columnstrict By default the last page is not set in a strict 'column-major' layout, if the logical pages do not fill up the whole page. The `columnstrict` option forces a strict 'column-major' layout for the last page. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `columnstrict=false`)

1	4	
2	5	
3		

`columnstrict=true`

1	3	5
2	4	

`columnstrict=false`

openright This option puts an empty page before the first logical page. In combination with `nup=2x1`, `nup=2x2`, etc., this means that the first page is on the right side. The same effect can be achieved with the `pages` option, if an empty page is inserted in front of the first page. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `openright=false`)

pagecommand Declares LaTeX commands, which are executed on each sheet of paper. (Default: `pagecommand={\thispagestyle{empty}}`)

turn By default pages in landscape format are displayed in landscape orientation (if the PDF viewer supports this). With `turn=false` this can be prohibited. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `turn=true`)

noautoscale By default pages are scaled automatically. This can be suppressed with the `noautoscale` option. In combination with the `scale` option (from graphicx) the user has full control over the scaling process. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `noautoscale=false`)

fitpaper Adjusts the paper size to the one of the inserted document. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `fitpaper=false`)

reflect Reflects included pages.

signature Creates booklets by rearranging pages into signatures and setting `nup=1x2` or `nup=2x1`, respectively. This option takes one argument specifying the size of the signature, which should be a multiple of 4.
An example for documents in portrait orientation:
`\includepdf[pages=-, signature=8, landscape]{portrait-doc.pdf}`
An example for documents in landscape orientation:
`\includepdf[pages=-, signature=8]{landscape-doc.pdf}`

signature* Similar to `signature`, but now for right-edge binding.

picturecommand Declares picture commands which are executed on every page within a picture environment with the base point at the lower left corner of the page. (The base point does not change if the page is rotated, e.g. by the `landscape` option.) (Default: `picturecommand={}`)

picturecommand* Like `picturecommand`, but with the restriction that `picturecommand*` executes its picture commands only on the very first page. (Default: `picturecommand*={}`)

pagetemplate By default the first inserted page will be used as a template. This means that all further pages are scaled to match within the contour of this first page. This option allows to declare another page to be used as a template; which is only useful if a PDF document contains different page sizes or page orientations. The argument should be a page number. (Default: `pagetemplate=`⟨*first inserted page*⟩)

rotateoversize This option allows to rotate oversized pages. E.g. pages in landscape orientation are oversized relatively to their portrait counterpart, because they do not match within the contour of a portrait page without rotating them. By default oversized pages are scale and are *not* rotated. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `rotateoversize=false`)

doublepages Inserts every page twice. This is useful for 2-up printing, if one wants to cut the stack of paper afterwards to get two copies. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `doublepages=false`)

doublepagestwist Whereas with `doublepages` the cutting edge is once on the inner side and ones on the outer side, `doublepagestwist` turns the pages such, that the cutting edge is always on the inner side. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `doublepagestwist=false`)

doublepagestwistodd Turns the pages such, that the cutting edge is always on the outer side. Either 'true' or 'false' (or no value, which is equivalent to 'true'). (Default: `doublepagestwistodd=false`)