Hans Hagen
pragma@wxs.nl

ConTEXt en pdf
Postprocessing pdf **files**
an application of TEXexec **and** pdfTEX

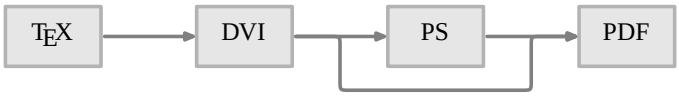
keywords
pdf, postprocessing, TEXexec, pdfTEX

abstract
This article introduces some ways to manipulate pdf files using pdfTEX, ConTEXt, and TEXexec. The method described here can be used for arbitrary pdf input, given that it can be handled by pdfTEX.

1 Introduction

The traditional TEX workflow can be summarized as follows:

A slight variation to this workflow is the direct conversion of DVI into PDF:



Both flows share that the intermediate formats DVI and POSTSCRIPT can be postprocessed with utilities, for instance to produce A5 booklets from A4 documents. Since TEX macro packages can use the `\special` primitive to add directives to the DVI file, effects not directly supported by TEX the program, can be achieved.

The previous chart shows a more direct way to produce PDF, the flow supported by PDFTEX. It will be clear that postprocessing now must take place at the PDF level.

Since PDFTEX can include pages from PDF files in a document, postprocessing can be handled by itself. In other words: PDFTEX can manipulate PDFTEX output. An advantage of this approach is that fonts are embedded efficiently. However, far more important is that one can use TEX to enhance the original documents while processing them again.

When postprocessing a PDF file, we can distinguish two categories: page imposition, which may lead to reordening of the pages, and collecting, which has a more linear nature.

I will limit the descriptions to the functionality as provided by TEXEXEC, the com-

mand line interface to CONTEXT. This does not mean that postprocessing is limited to files produced by CONTEXT: any reasonable and valid PDF file can be handled. TEXEXEC is only a PERL based wrapper, that generates the appropriate (relatively small) TEX files that do the job. By looking at the generated files `texexec.tex` one can get some insight in the CONTEXT commands involved.

2 Combinations

Especially presentations can be chacterized by an inefficient use of paper: relatively large fonts are used and the amount of text on a page is rather minimal. Therefore, when we want to print them, it makes sense to combine many of those pages on one sheet of paper. Such a page can be generated by saying:

```
\combinepages[pre-symb][nx=3,ny=5]
```

Of course it is not that convenient to key in commands like this for simple jobs, although a manual setup has the benefit that we can set more parameters than shown here. Using the default settings, TEXEXEC provides:

```
texexec --pdfcombine --combination=3*5 pre-symb
```

The small pages will be scaled in such a way that they comfortably fill the page. This is demonstrated on the next page. There are a few switches that controll the output:

`paperformat`	a predefined CONTEXT paper size, like `letter` or `A4`
`paperoffset`	a dimension specifying the margins in TEX units
`combination`	a `n*m` grid limited by the number of pages

3 Copying

Some printers, like ink–jet printers, have a relatively large unprintable area. The next command scales down a file so that it fits comfortably on the paper.

```
texexec --pdfcopy --scale=.95 yourfile.pdf
```

When one knows the unprintable margins, providing an offset makes more sense. The next call makes CONTEXT calculate the scale automatically:

```
texexec --pdfcopy --paperoffset=1.5cm yourfile.pdf
```

Both calls are especially useful when for instance the title page uses graphics (or color) that runs off the page.

`scale`	a (floating point) number like 0.85
`paperoffset`	a dimension specifying the margins in TEX units

4 Arranging

Say that one does not want to spend paper on printing the PDFTEX manual. Instead of printing he manual on A4, one can produce an A5 booklet.

```
texexec --pdfarrange --paper=a5a4 --print=up pdftex-a
```

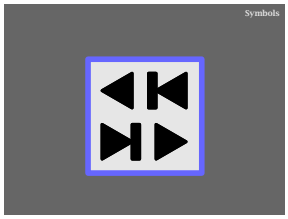
In this case it makes sense to add the following switch:

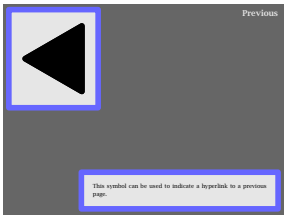
```
--addempty=1,2
```

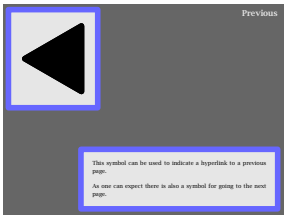
This directive tells CONTEXT to add two empty pages after page 1 and 2 (the title pages).

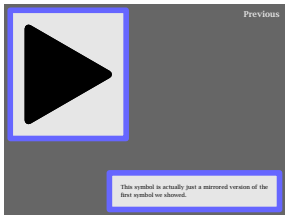
When asking for help (`--help pdfarrange`) one gets a list op options.

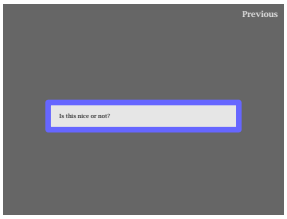
`paperoffset`	a dimension specifying the margins in TEX units
`paper`	a mapping like `a5a4` or `a4a3`

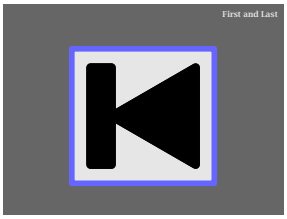

Symbols

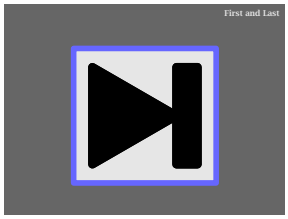

Previous
This symbol can be used to indicate a hyperlink to a previous page.

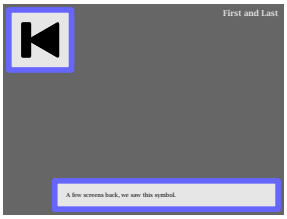

Previous
This symbol can be used to indicate a hyperlink to a previous page.
As one can expect there is also a symbol for going to the next page.

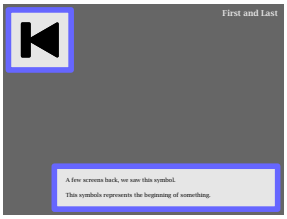

Previous
This symbol is actually just a mirrored version of the first symbol we showed.

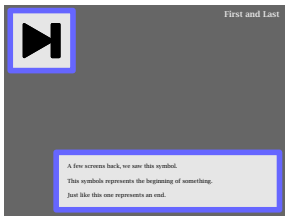

Previous
Is this nice or not?

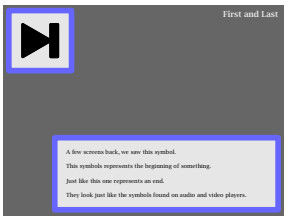

First and Last

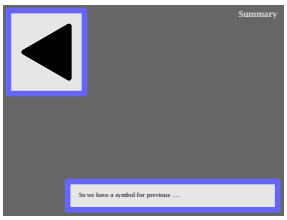

First and Last

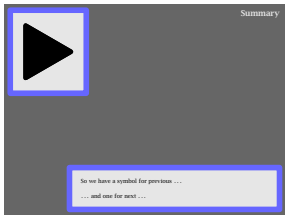

First and Last
A few screens back, we saw this symbol.

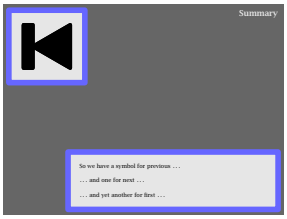

First and Last
A few screens back, we saw this symbol.
This symbols represents the beginning of something.

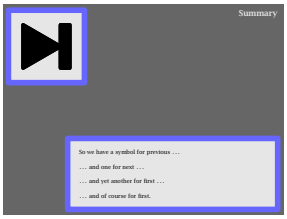

First and Last
A few screens back, we saw this symbol.
This symbols represents the beginning of something.
Just like this one represents an end.


First and Last
A few screens back, we saw this symbol.
This symbols represents the beginning of something.
Just like this one represents an end.
They look just like the symbols found on audio and video players.


Summary
So we have a symbol for previous ...


Summary
So we have a symbol for previous ...
... and one for next ...


Summary
So we have a symbol for previous ...
... and one for next ...
... and yet another for first ...


Summary
So we have a symbol for previous ...
... and one for next ...
... and yet another for first ...
... and of course for first.

`print`	an arrangement like `up` or `down`
`noduplex`	when issued, it forces single sided output
`backspace`	the side (inner) margin of the page in TEX units
`topspace`	the top (and bottom) margin of the page in TEX units
`markings`	when issued, cutmarks are added
`addempty`	a comma delimited list of pages after which to add an empty page
`textwidth`	the width of the original in TEX units (single sided)

In case of a single sided original with an asymetric layout, the width of the text should be specified to get the best results.

By providing more than one filename, one can combine files. This enables the user to add for instance a title and/or colofon page.

5 Selecting

One can create a stripped down version of a document by using `--pdfselect`. The next example filters some pages from two presentations and combines them into one document.

```
texexec --pdfselect  --paper=S6 --selection=1,3,8 --result=r-1 p-1
texexec --pdfselect  --paper=S6 --selection=2,5,9 --result=r-2 p-2
```

We can follow this up by:

```
texexec --pdfarrange --paper=S6 --noduplex --result=p-3 r-1 r-2
```

Again, there are some options:

`selection`	a list of pages to select, `odd` or `even`
`paperoffset`	a dimension specifying the argins in TEX units
`paperformat`	a predefined CONTEXT paper size, like `letter` or `A4`
`noduplex`	when issued, results in single sided output
`backspace`	the inner margin of the page in TEX units
`topspace`	the top margin of the page in TEX units
`markings`	when issued, add cutmarks
`addempty`	a list of pages after which to add an empty page
`textwidth`	the width of the original (one sided case)

6 Remarks

As I already pointed out, TEXEXEC's main task is to provide a proper command line interface to CONTEXT. Options are written to a option file, TEX is called with the CONTEXT format, and CONTEXT reads the options. When the job is finished, TEXEXEC calls TEXUTIL to sort the index, and, if needed, takes care of additional passes. Without changing the source file, one can invoke specific environments and style options, called modes.

Since its main task is to manage TEX runs, TEXEXEC can also be used to generate overviews of graphics, make listings of source code, generate module documentation, prepare formats, etc.

In the perspective of postprocessing PDF files the following option is worth mentioning:

```
texexec ...... --result=filename
```

By default, the results go to the file known as `\jobname`, which in the case of postprocessing PDF is `texexec.pdf`. The `--result` switch can be used to specify an alternative output file.

Another usefull option is `--help`, that can be followed by a switch specifier to get more help.

```
texexec --help pdfarrange
```

When using CONTEXT as macro package for processing TEX files, instead of arranging PDF pages, one can also rely on the built in page imposition mechanisms. These cover a rather wide range of possibilities and can be set up in the main document style. The `--noarrange` and `--arrange` switches control this process.

As demonstrated in a previous section, page imposition without the need to add directives to the document style is also possible. While the `--arrange` switch typesets the document at the requested size, the `--pdfarrange` option simply scales the pages and arranges them as images. Therefore:

```
texexec --arrange --paper=a5a4 --print=up somefile
```

and

```
texexec --pdfarrange --paper=a5a4 --print=up somefile
```

are fundamental different operations. The first one involves typesetting and moving pages around, the second concerns copying, scaling and moving of already typeset pages.

More information on these and other options can be found in the TEXEXEC manual. We expect to add more postprocessing features and options in the future. For more advanced and complicated cases one can always define a dedicated CONTEXT source file.

Since these facilities are still being extended and optimized, it makes sense to use the latest versions of PDFTEX, CONTEXT, and TEXEXEC. More information can be found at our home page: `www.pragma-ade.nl`, CTAN or one of the CONTEXTmirrors.