Typographers' Inn

Peter Flynn

Page numbering revisited

In my last column [2] I mentioned retrofitting page numbers from the PDF back into a web version of a document, and said it was relatively trivial with TEX. Daniel Nemenyi of KCL emailed me to ask how, so I had to dig out the code and see.

I know others have done this, but I don't know if anyone has documented it anywhere. What we implemented was for a client using XML, generating one transformation to XeLaTeX for creating PDF, and another to HTML for their in-house web site. The code is not proprietary but I can't extract it directly without exposing a lot of their in-house naming, so I rewrote a short version for Daniel.

The implementation was done using the fwlw package (which makes catchwords available for each page: the first and last words on the page plus the first word of the next page). With this, we modified the `\pagestyle` provided to typeset the catchwords at the bottom of the page, in white, so they were not visible, and separated them by an otherwise unused delimiter so we could extract them reliably: we used the ASCII decimal 172 (0xAC) character (¬) or `NOT` symbol (see Figure 1).

```
\documentclass{article}
\usepackage{lipsum,fwlw}
\usepackage{xcolor}
\makeatletter
\def\ps@pagerange{\let\@mkboth\@gobbletwo
 \let\@oddhead\@empty\let\@evenhead\@oddhead
 \def\@oddfoot{\rlap{\color{white}%
     Page=\thepage¬First=\usebox\FirstWordBox¬%
     Last=\usebox\LastWordBox¬%
     Next=\usebox\NextWordBox¬}%
       \hfil\thepage\hfil}%
 \let\@evenfoot\@empty
 \let\chaptermark\@gobble
 \let\sectionmark\@gobble
 \let\subsectionmark\@gobble
}
\makeatother
\pagestyle{pagerange}
\begin{document}
\lipsum[1-100]
\end{document}
```

Figure 1: Minimum worked example to expose catchwords for retrieval.

For the extraction we used *pdftotext*, a freely-available utility which creates a plaintext version of a PDF document. In this, page-breaks are signalled with an ASCII decimal 12 (0x0C) character, which is the Control-L or FF (FormFeed). In this example, the few lines immediately above each of the page-breaks contains the page number preceded by the delimited string we defined in `\ps@pagerange` in Figure 1.

In Figure 2 you can see two fragments of the output, the first from page 1 and the second from page 16 showing a problem where the page number occasionally gets imbrangled in the 'Next' catchword. This has not been resolved.

```
lorem lorem, interdum eu, tincidunt sit amet,
Page=1¬First=¬Last=amet,¬Next=laoreet¬
1

^Llaoreet vitae, arcu. Aenean faucibus pede eu
ante, Praesent enim elit, rutrum
```

```
tempus magna. Aliquam ut purus. Proin tellus.
Page=16¬First=amet,¬Last=tellus.¬Next=
16

Vestibulum¬

^LVestibulum ante ipsum primis in faucibus
```

Figure 2: Text fragments of output from Figure 1 at pages 1 and 16.

Now that the data is plaintext, you can use the standard *grep* and *awk* text utilities (or Perl, or Python, or Lua, or whatever is your favourite scripting language *du jour*) to pull out the lines with the delimited page number, first, last, and next words. You can then programmatically step through each page number and locate the span of text delimited by the First and Last words, using the Next word as a cross-check.

The tricky bit is application-dependent: you then need to be able to reliably read your source text programmatically, find the first word on a page, scan forward to the last, check the following word is the next value, and then do whatever is needed to insert the page number at whatever point is appropriate for your document.[1]

In the case in point, the production text was stored as XML, so the delimiters they used for the line of data embedded in the PDF were actually < and > characters, so the extracted fragments were already XML. That way the lines extracted from the text file were used in XSLT to identify each location in the XML source, push the page numbers into

[1] Daniel did suggest it might be more tractable to write the page-break data to a separate external file rather than embedding it: I'd be interested to hear from anyone implementing this.

doi.org/10.47397/tb/42-1/tb130inn

a Processing Instruction, and cyclically check the accuracy each time the file was processed. If the source is LaTeX, it might be more problematic to process.

It's not 100%, of course: it will be thrown by figures, tables, and math occurring at a page boundary, which our client didn't use. But the small number of corrections beats doing it by hand.

Type 1 (PostScript) fonts

Some of you may by now have seen Adobe's announcement[2] that it will end support for Type 1 (PostScript) fonts on 31st January 2023 in all its products (e.g. *InDesign*, *Illustrator*, *Photoshop*, etc.).

If you open a document containing unembedded Type 1 fonts with an Adobe product after that date, the fonts will not be recognised, and will be classed as 'Missing' even if you have the font files installed in your operating system. In addition, your installed Type 1 fonts will no longer appear in the Fonts menu and there will be no way to use them in Adobe's software.

However, PostScript and PDF documents with embedded Type 1 fonts will continue to display as normal, so they will still be readable with Acrobat Reader, but they will not be editable and will not work in other Adobe products.

So what's this all about? To be fair, Type 1 fonts are Adobe's invention (back in 1984), so they can call the shots. When PostScript printers arrived, they came with the built-in Adobe '35' popular fonts[3] that have dogged DTP ever since (a much wider choice was distributed later). Those 35 fonts became so ingrained that software producing PostScript (and later, PDF) only needed to reference the font by name, with no need to embed it in the document, because it could be guaranteed to be available on all printers via drivers like *Ghostscript*. It's also why so many packages that create formatted output, like statistical and numerical analysis programs, generate PostScript and PDF output without the need to embed the fonts. Plus they were seen as 'free' — in a world where font piracy is rampant, many users became accustomed to the idea that [these] fonts 'just came with' every operating system and software suite.

But the world has moved on since then, and font technology has advanced hugely. TeX has moved on too, from providing only Computer Modern and other METAFONT fonts (Type 3), to updated Type 1 versions as well as the Adobe '35' and other Type 1 fonts generously donated to the cause. You could buy or download additional Type 1 fonts and install them for use with TeX. And now we can use any OpenType or TrueType font via XeTeX, LuaTeX, and friends, including the TeX Gyre fonts (open source equivalents of the Adobe '35').

Does it matter to us? Well, yes ... some. We need to be aware that in the long term, Type 1 is going to become a dead end. For now, if you have old PostScript or PDF documents for which no source is available, they will continue to display. If you want to continue generating PostScript or PDF documents using the Type 1 fonts that come with TeX distributions, or others you have bought or downloaded, feel free to do so: your output documents will continue to be displayable. TeX itself is unaffected, and so far as I have been able to find out, neither is software from other suppliers, so you can continue using Type 1 fonts in TeX, and in many other non-Adobe systems.

So what's to do? The easy answer is, switch to XeTeX or XeLaTeX or LuaTeX or one of the other variants that support OpenType or TrueType fonts. I made that switch a couple of years ago and have not regretted it [1]. But there are still a lot of LaTeX packages that depend on PostScript fonts or graphics for other reasons, and if you use them, you may need to stick with *pdflatex* for a while yet.

Afterthought

I couldn't trace the quotation 'There is not in existence a page with a rule on it that cannot be instantly and obviously improved by taking the rule out.' [3] but Karl Berry pointed me at *The TeXbook* (end of Chapter 21). I should have looked there first!

References

[1] P. Flynn. Typographers' Inn — XeLaTeX. *TUGboat* 37(3), Sep 2016. tug.org/TUGboat/tb37-3/tb117inn.pdf

[2] P. Flynn. Typographers' Inn — To print or not to print. *TUGboat* 41(3), Dec 2020. tug.org/TUGboat/tb41-3/tb129inn.pdf

[3] G.B. Shaw. On Modern Typography. *The Dolphin: A Journal of the Making of Books* 4(1):80–81, Fall 1940.

◇ Peter Flynn
Textual Therapy Division,
Silmaril Consultants
Cork, Ireland
Phone: +353 86 824 5333
peter (at) silmaril dot ie
blogs.silmaril.ie/peter

[2] https://helpx.adobe.com/ie/fonts/kb/postscript-type-1-fonts-end-of-support.html

[3] These are: Avant Garde, Bookman, Courier, Helvetica, New Century Schoolbook, Palatino, Σψμβολ (Symbol), Times New Roman, *Zapf Chancery*, and Zapf ✤✻■✳❁❁▼▲ (Dingbats).