How to make a product catalogue that doesn't look like a dissertation

Jason Lewis

Abstract

Needing a robust way to produce a catalogue from a product database, Adobe InDesign, DocBook and LaTeX were evaluated. LaTeX was favoured due to its layout flexibility and non-proprietary nature. The challenge was to query the database and produce suitable LaTeX for the catalogue

Native LaTeX was unable to query the database, so templating tools were investigated. Template Toolkit (TT) was chosen over PHP, favouring its Perl roots and broader applicability. Using TT and `DBI` for querying the database a dynamically generated LaTeX document can be quickly constructed. TT filters are developed to check and correct user supplied content for constructs that would cause the LaTeX compiler to fail. Filters are also used to sanitise Windows file paths for use in LaTeX.

Styling the document to look like a product catalogue was achieved using sans serif fonts, `flowfram` thumb tabs, colourful chapter and section headings developed using `tikz`, long tables that allow page breaks, alternating row colours, wrapping of paragraph text around images and the highlighting of new products. Full page PDFs are included in the document for the covers, front matter and adverts.

The result is system to generate a product catalogue, quickly and easily directly from our database using free and open source tools. This has reduced the workload in producing a catalogue and increased staff productivity and efficiency.

1 Introduction

I produce an 80 page full colour product catalogue in LaTeX. This paper outlines the tools I used to implement the system, link it to our database and style the catalogue for printing.

We print a new catalogue every six months and it has approximately 1000 products, 16 full page colour adverts and takes less than two minutes to build from the command line.

2 Why did I write this?

I am part owner of a small wholesale distribution business in Australia. When we established the business in 2001, we had approximately 40 products in our catalogue, for which a one page price list and order form in Microsoft Excel sufficed.

As we grew the business and added more products to our portfolio, the catalogue became unwieldy to produce in this way. So in 2004 I set out to create a more robust system to produce it.

My goals were to create a system that would produce a product catalogue automatically from our database, wouldn't require long and detailed proof reading to ensure pricing was correct, be something I could delegate to staff for catalogue production, and that the staff need not have any special technical knowledge to use the system.

2.1 Tools I looked at

First, InDesign:

- InDesign at the time had limited scripting ability. This has improved now and you can write scripts in Microsoft Visual Basic or JavaScript.
- no proper database connectivity at the time. There are now numerous commercial tools that allow you to link InDesign to a database.
- proprietary software (I prefer to use free tools if possible).

Second, DocBook [1]:

- limited formatting and layout capabilities
- recommends using LaTeX for advanced layouts
- no database connectivity
- plain text, easy to script using a template tool

Third, LaTeX:

- flexible layout capabilities
- free and open source
- no database connectivity
- plain text, easy to script using a template tool

3 The challenge

3.1 How to script LaTeX?

I could have written the system in native LaTeX but LaTeX has no way to retrieve data from a database. The other option was to choose a tool like PHP or Template Toolkit (TT).

3.2 PHP

Pros: PHP is widely used.

Cons: geared towards HTML/web; it's PHP; it's not Perl.

3.3 Template Toolkit

Pros:

- Generalised to templating anything
- not PHP
- written in Perl
- `Template::Plugin::Latex` can output PDF directly; can build manually for demonstration.

Astute readers might notice a slight bias towards Perl. This was mainly due to having prior knowl-

edge of Perl and meant I didn't have to learn a new programming language.

3.4 Database driven documents

Template Toolkit uses Perl's DBI for database access, which means it can retrieve data from just about anything. Our data is stored in Microsoft SQL and Microsoft Access. I use DBI Proxy [2] to connect to the MS Access database.

3.5 Compiling reliably

Building the catalogue from source is a two step process: running Template Toolkit, and then passing its output through `pdflatex`. I created a Makefile to do this, but often the document would require multiple runs to ensure all the cross-references were correct. A tool such as `latexmk` [3] or `rubber` [4] helps with this by providing a single command that will repeatedly run `pdflatex` until all cross-references are stable.

3.6 User interface for entering data to the catalogue

I needed a user interface for data entry. I chose MS Access as being easy to script, it was a familiar user interface for the staff, and I already owned it. The downside is that of course it is a proprietary tool.

4 Template Toolkit Primer

Here's how I scripted LaTeX using Template Toolkit.

4.1 Install Template Toolkit

Cpanminus [5] is a great tool to retrieve, unpack, build and install Perl modules from CPAN:

```
cpanm Template
cpanm DBD::CSV
cpanm DBI
cpanm Template::Plugin::DBI
```

Now we have Template Toolkit, `DBI`, and some ancillary modules installed, and can try writing a simple Hello World.

4.2 Hello World

`tpage` is a simple script supplied by TT that parses a template file supplied on the command line and outputs it to standard output. Given this two-line template file:

```
[% str = 'Hello TUG2013' -%]
[% str %]
```

We can parse and output it like this:

```
$ tpage helloworld.tt
Hello TUG2013
```

Anything within `[% %]` will be treated as TT code and executed. The minus sign before the closing delimiter above strips the final newline from that line upon output; similarly, a minus sign after an opening delimiter (as below) removes a leading (preceding) newline.

4.3 Read a CSV file

Let's suppose we have this table of data:

Table 1: `simple-example.csv`

FirstName	LastName	FavouriteNumber
Jason	Lewis	34
Joe	Blogs	2
Frank	Sinatra	88

Here is TT code to parse it, using `DBI` to access the file as a database in the current directory:

```
[%- USE db = DBI(
   database => "DBI:CSV:f_dir=." ) -%]
[%- FOREACH item = db.query(
   "SELECT * from simple-example.csv")  -%]

FirstName: [% item.firstname %] \
LastName: [% item.lastname %] \
Favourite: [% item.favouritenumber %]
[% END %]
```

Note that the column headings of table 1 were sanitised to lower case by `DBI::CSV`.

4.4 Write your own parser

Well not quite, but Template Toolkit makes it easy to implement a TT parser:

```
#!/usr/bin/env perl
use Template;
die "no TT filename given" if (@ARGV != 1);
my $tt = Template->new({
    INCLUDE_PATH => '.',
    INTERPOLATE  => 1,
}) || die "$Template::ERROR\n";

my $input = $ARGV[0];

# process input template, substituting variables:
$tt->process($input, $vars)
  || die $template->error();
```

This creates a TT object and parses the file whose name is supplied on the command line. So far, this just provides the same functionality as `tpage`.

4.5 Build your LaTeX document

Here we write our LaTeX document, including the TT lines to read from the database.

```
[%- USE db = DBI( database =>
             "DBI:CSV:f_dir=." ) -%]
\documentclass{article}
```

```
\usepackage[utf8]{inputenc}

\title{build-document example}
\author{Jason Lewis}
\date{October 2013}

\begin{document}
[%- FOREACH item = db.query(
   "SELECT * from simple-example.csv")  -%]
First Name: [% item.firstname %]
\\ Last Name: [% item.lastname %]
\\ Favourite Number: [% item.favouritenumber %]
\\ \newline
[% END #FOREACH -%]
\end{document}
```

Output (where `mytpage` is the script we just saw):

```
$ ./mytpage build-document.tt
\documentclass{article}
\usepackage[utf8]{inputenc}

\title{build-document example}
\author{Jason Lewis}
\date{October 2013}

\begin{document}
First Name: Jason \\ Last Name: Lewis
  \\ Favourite Number: 34 \\ \newline
First Name: Joe \\ Last Name: Blogs
  \\ Favourite Number: 2 \\ \newline
First Name: Frank \\ Last Name: Sinatra
  \\ Favourite Number: 88 \\ \newline
\end{document}
```

5 Technical problems

5.1 Escaping special LaTeX characters

The text in our database is generated by users, and often cut and pasted from Microsoft Word. It typically contains characters that won't be natively typeset by LaTeX and worse still, cause the LaTeX build to hang or fail.

Therefore all text from the database needs to be sanitised before being passed to LaTeX. My solution was to write a TT filter. Below is a sample function that takes a string as its input, and ensures any dollar signs within the string are escaped by prepending a backslash.

5.2 Sanitise for LaTeX

```
sub latex_filter {
  my $return = $_[0];

  # escape $ with a backslash for LaTeX
  $return =~ s/(\$)/\\$1/g;

  return $return;
}
```

```
my $tt = Template->new({
  FILTERS  => {latex_filter => \&latex_filter},
  INCLUDE_PATH => '.',
  INTERPOLATE  => 1,
}) || die "$Template::ERROR\n";
```

5.2.1 More things that need to be filtered

Then adding more filters is simply a matter of adding a regular expression to search for it and replace it with whatever is appropriate. Some examples.

Degree symbol:

```
$return =~ s/°/\\degree /g;
```

Accented characters: Transform é into `\'{e}`:

```
$return =~ s/é/\\'{e}/g;
```

Incorrect quotes: Convert beginning-of-line right or double quotes to left quotes, and use ASCII apostrophes.

```
$return =~ s/(^|\s)'(.*?)/$1‘$2/g;
$return =~ s/(^|\s)"(.*?)/$1‘‘$2/g;
# replace Windows quote (octal 0222) with ASCII '
$return =~ s/\222/'/g;
```

En dash between numbers: Convert minus sign to an en-dash by searching for single minus sign between numbers and replacing it with two minus signs: `1-10` vs. `1--10`.

```
$return =~ s/(\d+)-(\d+)/$1--$2/g;
```

5.3 Pass through LaTeX commands

From time to time I needed a way to pass LaTeX commands through the filter. I chose an escape of `<latex>` which in hindsight may not have been the best choice, as it looks too much like XML. We just replace it with a backslash:

```
$return =~ s/<latex>/\\/g;
```

For example, `<latex>latex` becomes `\latex`.

5.4 Calling a filter from within a template

Once a filter has been defined, you can call it with the pipe '`|`' character.

```
[% str = "$100 is 50% of $200" -%]
Raw string is [% str %]
Filtered string is:
   [% "$100 is 50% of $200" | latex_filter %]
```

Which results in:

```
Raw string is "$100 is 50% of $200"
Filtered string is:
    "\$100 is 50\% of \$200"
```

5.4.1 LaTeX does not like Windows paths

On our network, product images are stored on a server drive accessed via a Windows share `w:\`. In MS Access, users select an image to go with a product

range and the path to the image is stored in the database as a Windows path:

```
w:\some\path to\an image.jpg
```

Users often use spaces, commas, apostrophes and other abominable characters in the image paths, and `\includegraphics` does not handle paths with spaces in them.

The solution was to create a LaTeX-friendly symbolic link to the file and include that instead. This was easy to achieve by making another TT filter.

5.4.2 Convert Windows path to Unix

The goal is to convert a Windows path such as

```
w:\Alive & Radiant\2013-July-product.jpg
```

to:

```
_mnt_Alive_&_Radiant_2013-July-product.jpg
```

5.4.3 Build a filter

Here is my Perl code:

```
# convert all \ to / e.g. c:\ to c:/
$return =~ s/([\\])/\//g;
# strip drive letter c:/path/file to path/file
$return =~ s/^\w:\///g;
# strip extension: /filename.jpg to /filename
#$return =~ s/\.\w\w\w$//g;
# clean up the path
$return = File::Spec::Unix->canonpath($return);
```

5.4.4 Make the symbolic link

Find spaces, / or % in filenames and replace with underscores. Then use the resulting string as the name for a symbolic link to the original file. Use the symbolic link name in the LaTeX document.

```
# replace spaces, slashes, percents with _
$safe_filename =~ s/[\s\/%]/_/g;
# make a symbolic link to the file
symlink($return, $safe_filename) || die;
```

5.4.5 Use new filter on image paths

Use the new filter on image paths as they are retrieved from the database:

```
[% ImagePath
 = item.CategoryImagePath | path_filter %]
[% IF ImagePath != "" %]
    \includegraphics[width=12cm,
                     height=\imageheight,
                     keepaspectratio=true]
                     {[%ImagePath%]}
[% END # if image exists %]
```

5.5 User interface

I needed to develop a user interface for staff to manage data in the catalogues. I chose MS Access as it was easy to create and modify. I wanted to create something that would shield the users from having to know LaTeX in order to create and edit content for the catalogue.

6 Make LaTeX output look like a product catalogue

Clearly I had to style the document so it would look more like a product catalogue and less like a dissertation.

6.1 Sans serif fonts

The first thing I did was use a sans serif font. I chose Helvetica.

```
\usepackage{helvet}
% set the font to helvetica for body text
\renewcommand{\familydefault}{\sfdefault}
```

This provides URW Nimbus Sans which is a free clone [6] of Helvetica.

6.2 Thumb tabs

Thumb-tabs are a nice feature for any catalogue. I created them using Nicola Talbot's `flowfram` [7] package, like this:

```
\setthumbtab{1}{backcolor=[rgb]{0.15,0.15,1}}
\setthumbtab{2}{backcolor=[rgb]{0.2,0.2,1}}
\makethumbtabs[50mm]{30mm}

\begin{document}
  \tableofcontents
  \thumbtabindex
  \enablethumbtabs
  \chapter{1ABC}
  \Blindtext
  \chapter{2ABC}
  \Blindtext
\end{document}
```

6.3 Colourful chapter and section headings

Colourful chapter and section headings were made using the `tikz` package.

```
\newcommand\colorchapter[1]
  {\def\chapterbg{#1}\chapter}
% begin CHAPTER format
\newcommand\boxedchapter[1]{{%
  \begin{tikzpicture}[inner sep=0pt,
                      inner ysep=1.3ex]
% left position of text
% right hand edge chapter title text
  \node[anchor=base west]
    at (3,0) (counter) {};
  \path let \p1 = (counter.base east)
    in node[anchor=base west,
    text width={\textwidth-\x1+26.33em}] (content)
    at ($(counter.base east)+(0.33em,0)$)
    {\textcolor{white}
     {\Huge\sffamily\textsc{\thechapter \ \ #1}}};
```

```
  \begin{pgfonlayer}{background}
   \shade[left color=\chapterbg,
          right color=\chapterbg]
    let \p1=(counter.north),\p2=(content.north)
        in (0,100 + \maxof{\y1}{\y2})
        rectangle (content.south east);
  \end{pgfonlayer}
\end{tikzpicture}%
}}

\titleformat{@@html:&#92;@@chapter}%
    {}%
    {}%
    {0pt}%
    {\boxedchapter}%
    \titlespacing*{\chapter}{-100pt}{*-20}{*-1}
% end CHAPTER format
```

6.4 Long tables

I needed a way to create tables that could break across page boundaries, but also be able to span more than one page, and possibly have a PDF (for an advert) embedded at a split. The `xtab` [8] package produced the best results for me; however, there are very many packages for tables. I found a good summary of table package features at `http://tex.stackexchange.com/a/12673`.

6.5 Alternating row colours in `xtab`

This was easy to do in TT: while looping through query results, simply change the row colour every two lines.

```
\begin{xtabular}[l]
  [% i = 1 -%]
  [% FOREACH item = db.query(
      "select * from $CatInfo.query") %]
    [%- IF (i mod 4 == 3)
          || (i mod 4 == 0) -%]
        \rowcolor{[%-
            item.SectionRow2BGColour -%]}
    [% ELSE -%]
        \rowcolor{[%-
            item.SectionRow1BGColour -%]}
    [%- END #iF -%]
    [%- i = i+1 -%]
    [% item.col1 %] &
    [% item.col2 %] &
    [% item.col3 %] \\ %row data
  [% END; #FOREACH %]
\end{xtabular}
```

6.6 Wrap description around an image

I used `wrapfig` [9] to put text around an image.

```
[% IF ImagePath != "" %]
  \begin{wrapfigure}{r}{0pt}
    \includegraphics[width=12cm,\
                     height=\imageheight,\
                     keepaspectratio=true]\
                     {[%ImagePath%]}
  \end{wrapfigure}
[% END # if image exists %]
```

6.7 Highlight new products

New products are highlighted by yellow text with a red background.

```
[%- IF item.NewProduct -%]
  \scriptsize\colorbox{red}
  {\textcolor{yellow}{NEW}}
  \sffamily\footnotesize{~ \
  [%-item.description | latex_filter -%]} &
    \footnotesize{[%-item.description |
    latex_filter -%]} &
[%- END -%]
```

6.8 Including full page PDFs

We include full page PDFs in the catalogue for the front matter, rear matter and adverts. All are supplied as PDFs and we just have to include them. The trick is to turn off scaling so the bleed and trim marks appear in the correct place.

```
[% IF String.length > 0 %]
  \includepdf[noautoscale=true]{[%CoverPage%]}
[% ELSE # warn that file cannot be found %]
  Cover file [%  CatInfo.CatalogueFrontPage %]
      appears to be missing : [% error.info %]
[% END %]
```

7 Conclusion

I set out to create a tool for generating a catalogue from our database. I was able to use free tools such as LaTeX and Template Toolkit to achieve this. The overall goal was achieved, saving time and money and allowing staff to be more productive.

References

[1] `http://www.docbook.org/docbook`
[2] `http://search.cpan.org/dist/DBI/dbiproxy.PL`
[3] `http://ctan.org/pkg/latexmk`
[4] `http://launchpad.net/rubber`
[5] `http://search.cpan.org/dist/App-cpanminus/bin/cpanm`
[6] `http://www.tug.dk/FontCatalogue/helvetica`
[7] `http://ctan.org/pkg/flowfram`
[8] `http://ctan.org/pkg/xtab`
[9] `http://ctan.org/pkg/wrapfig`

◇ Jason Lewis
`http://organictrader.com.au`