The PracTEX Journal, 2010, No. 1
Article revision 2010/05/27

Automatic report generation with your text editor, Perl, and LaTeX

Richard Hardwick

Email rch@skynet.be
Website http://users.skynet.be/watermael/home.html
Address Auderghem, Belgium

Abstract I describe a simple system for producing standard evaluation reports. The evaluator writes plain text files. A Perl script reads the text files and uses the Perl module "Template.pm", with a ready-made LaTeX template, to generate the final LaTeX report.

1 Introduction

Boris Veytsman and Maria Shmilevich have eloquently described [1] the traditional way by which contractors produce the reports demanded by funding agencies -

The contractor's employees send e-mails to their managers describing their accomplishments. A manager copies and pastes these data into a Microsoft Word file and sends the file to the next level manager, who collates the received reports together. The task is repeated regularly, and each piece of information is copied and pasted several times: in weekly, monthly, quarterly and yearly reports. If the contract involves many tasks and subtasks, the work is overwhelming. This is unproductive work, since the real task of managers is management, not copying and pasting repetitive chunks of text.

Veytsman and Shmilevich describe a system with Web, TEX and SQL which enables the routine aspects of these tasks to be automated, so that the report is quickly and efficiently completed and delivered. In a phrase, their system separates content from form.

1. Automatic report generation with Web, TEXand SQL. Boris Veytsman and Maria Shmilevich (2007) TUGboat 28:1, pp 77-70 http://www.tug.org/TUGboat/Articles/tb28-1/tb88veytsman-report.pdf at http://www.tug.org/TUGboat/Contents/contents28-1.html

2 Evaluation reports

The story of *unproductive work* does not end there. Once the contractor's report is received at the funding agency, it has to be read and evaluated. Just as learned journals send papers out to referees, so funding agencies send reports out to external assessors. Nowadays most journals, and some funding agencies, ask their referees to complete an evaluation page at a password-protected website with a CGI interface. But other funding agencies still just send a form, usually in Microsoft Word format, which the assessor has to fill in and send back by email. Not only does this require *... copying and pasting repetitive chunks of text ...* but bitter experience also shows that copying and pasting may awaken previously hidden formatting commands. The result will be the phenomenon of WYDSCCYG (WHAT YOU DON'T SEE CAN CAUSE YOU GRIEF), not unknown to users of Microsoft Word.

I have been struggling with the evaluation of annual reports for an Agency which funds botanical research. My initial attempts to fill in the blanks in their document using a WYSIWIG editor revealed many instances of WYDSCCYG. I was much more productive writing my comments in ASCII, and then pasting them into the evaluation document. But this seemed ridiculously unprofessional, even for a botanist. Then I came across Veytsman and Shmilevich's article and I was inspired to try to follow their example. I thought that the job would be simple - in order to separate content from form, put the content in an ASCII text file, put the form in a LaTeX template, and use a little Perl script to put form and content together again - to read the ASCII and automagically fill in the template's blanks. However, I have little knowledge of Perl, and even less of LaTeX, so it became a learning experience.

2.1 How to go from Microsoft Word to LaTeX

The evaluator is required to send back to the funding agency a duly completed 'copy' of the agency's original document. So the first challenge was to write LaTeX code that would reproduce as exactly as possible the appearance of the Agency's ''Evaluator's guide".

After a few miserable attempts to make a fair copy in LaTeX and PDF of a document that had been lovingly created (and many times editted) in Microsoft

Word, OO.org came to my rescue. I discovered that if you open a Microsoft Word document in OpenOffice.org, press the button “File/Export", and choose “File format LaTeX2e", the job is done in milliseconds.

So now I had my LaTeX template. Running the template through pdfLaTeX produced a PDF document that was a satisfyingly close approximation to the agency’s original assessment form. The Agency would get their documents back in good shape. It just remained to fill in the blanks in the LaTeX file. But then a quick look at the LaTeX revealed a jumble of “`\textsf`" and “`}}`"’s, and worse:

```
{\selectlanguage{english}
\textsf{\textbf{Main activities and interim results achieved:}}\textsf{
}}
\item {\selectlanguage{english}
\textsf{Describe in a concise style the main achievements, interim
results and deliverables of the WP during the reporting period.}}
```

Much of this appeared to be unnecessarily repetitive, and some of it was obviously redundant. It looked as though the Agency’s document had been through a lot of edits. But before I could use the generated LaTeX as a template I needed to cut my way through that thicket of “`\textsf`" and “`}}`"’s. Easier said than done; it was all too easy in chopping out some redundant command to miscount the opening and closing brackets, and to leave the brackets unbalanced. I quickly tired of the LaTeX error message -

```
! Missing } inserted.
<inserted text>
                }
l.850 \end{itemize}
```

and friends. My solution, after some head-scratching, was to write a tiny Perl utility which runs through the LaTeX file and prints it out again line-by-line, together with a cumulative line-by-line count of the opening { and closing } brackets. The Perl script looks like this . . .

```
while(<FILE>){
    if(/^\%/){          # lines beginning "%" go straight to output
        print STDOUT;
    }
    else{               # any other line; count the brackets
        $_ =~ s/\%.+$//;# first discarding everything after a "%"
        $opening_count += s/\{/\{/g;
        $closing_count += s/\}/\}/g;
        $status=$opening_count - $closing_count;
        print STDOUT "$status \t $.\t $_";
    }
}
```

A long run of "0"'s in the output indicates that brackets were balanced - at least *that* problem should have been solved. A long run of positive numbers means that there is one or more { too many, a run of negative numbers indicates a surplus of }'s. With this script to help me it became much easier to pinpoint the problems in the LaTeX file. After several iterations of ... edit LaTeX/ run script / edit LaTeX/ run script / compile LaTeX/ ..., I had a relatively clean LaTeX file, my candidate for the template.

2.2 How to fill in the LaTeX Template, using Perl

It now only remained to make an ASCII file of evaluator's comments, and to use that to fill in the appropriate gaps in the LaTeX template. A neat little Perl script should do the trick. But here I met another complication. The form that the Agency sends out to their evaluators has standard boiler-plate texts for just one "work package", one "task", and one "sub-task". But of course any real-life project comprises several "work packages", each of those work packages has several "tasks", and each task has several "sub-tasks". If you are working in Microsoft Word, that means a lot of cutting-and-pasting of work package, task, and subtask boiler-plates. It should be easy to do the job with a Perl script. We are just going to need three levels of `foreach` loops and a corresponding three-deep hash-of-hashes-of-hashes. The number of parent-, of children-, and of grandchildren-nodes is not known in advance. In Perl that is no problem; Perl

structures magically grow as necessary. But how to match that flexibility in the LaTeX template? I was beginning to get out of my depth, again.

I finally discovered a nice Perl package; Template.pm [2], which allows the full gamut of Perl logic to be implemented inside the LaTeX template. In Perl, to print out subtask information, you might write set of foreach loops, the first one looping over work packages, the second looping over tasks within work packages, the third looping over subtasks within tasks. With Template.pm you can do the same in your template. You can embed all the loops and logic in your LaTeX template, like this ...

```
[% FOREACH WPnum IN Task.keys.sort -%]
     [% FOREACH Tasknum IN Task.$WPnum.keys.sort -%]
       Activities and results:\\
       [% Task.$WPnum.$Tasknum.TaskActivitiesAndResults %]\\
    [% END %]
[% END %]
```

You pass Template.pm a Perl hash with entries such as

```
$hash{Task}->{1}->{2}->{TaskActivitiesAndResults}=
 '17 accessions have been added ...';
```

and it reads the hash, fills in the values in your template and makes the final LaTeX file with the right numbers of Work packages, Tasks and SubTasks, and all the blanks nicely filled in. Run pdfLaTeX on that file, and your report is ready to send to the funding agency. As proof of concept I can even use the hash from an evaluation I have just completed, with the LaTeX source of this very article as template. The output is a file, which starts and finishes as the LaTeX source, but in which the lines above, beginning

```
[% FOREACH WPnum IN Task.keys.sort -%]
```

etc
have disappeared. They have been replaced by

2. http://template-toolkit.org/

```
     Activities and results:\\
Partners collected a total of 145 accessions of blue-green algae ...
    Activities and results:\\
17 accessions have been added from Iran and Afghanistan ...
    Activities and results:\\
Trials were undertaken with four species ...
...
```

Those are indeed my evaluations of the various activities and results of the Alga project. Template.pm has recognised the lines "[% ... %]" as Template.pm code. It has reacted by filling in the appropriate values from my hash, task by task and work package by work package. [3]

One last problem - how to get the values into the hash?

3 How to fill in the hash

I just counted the elements that have to be completed in one of my Agency's evaluation forms. There are 316 such elements. 182 of them are items such as the coordinator's name, e-mail address, fax, and telephone number, the name of each work package, each task, and of each subtask. The values of all these items are already available, in the coordinator's report which I have to evaluate. So I wrote a Perl script to pull these values out of the report and store them in the hash. That left 134 items that have to be judged by the evaluator ("AchievementsAllWPs", "BudgetEvaluation", ..., ..., "Twelvemonthswhatprogress"). Veytsman and Shmilevich have the most elegant approach to capturing these values - the text is input straight into a relational database. That technology is a bit beyond

3. Observant readers may be wondering what really happens when I run Template.pm on the LaTeX source of this present article. There seem to be two Template.pm FOREACH statements in the current text. Doesn't the second FOREACH cause Template.pm to complain about an unbalanced loop? The answer is yes, it did, until I thought of adding a second [% END %] right here. (Actually, there are two [% END %]'s right here; the [% END %] statement that keeps Template.pm happy (but which you cant see because it is commented out by a % because LaTeX doesnt seem to like *verbatim* commands in footnotes), and another [% END %] statement so that you can see it (but which is ignored by Template.pm, because it is actually coded [\% END %\], for LaTeX). Thats the sort of contorted thinking that you find yourself having to do, when you mix LaTeX with other languages.)

my capacities, so I again settled on ASCII files. The evaluator gets one such file for each work package, plus one for the overall evaluation of the whole project. Each file is made of paragraphs copied from the Agency's evaluation form, explaining what is needed from the evaluator regarding e.g. Task activities and results. The final line of each paragraph is the key, an "=" sign, and a blank space - like this:

```
!TaskActivitiesAndResults =
```

Using emacs, Vim, or whatever, the evaluator fills in the blanks with lines of text (as many as you like). Then Perl does the rest. The lines written by the evaluator become the values of the keys, Perl passes the hash to Template.pm, and then (if all has gone well) runs pdfLaTeX on the output. The Agency gets the PDF by return.

4 Conclusion

Evaluators should devote all their attention to the words on the page; i.e. to the content of their report. They should not have to bother their heads about the form of their report. The best way of providing that form is undoubtedly something like Veytsman and Shmilevich's automatic report generator. But that requires a web server, a knowledge of SQL, and a certain investment of time and resources. Here I have described a poor man's alternative. And as a poor man I have discovered that when you need some string, or some sealing wax, the Perl community, and the TEX community, have that in abundance. So I close with big thank-yous to Henrik Just for his *W*riter2LaTeX[4], which is an now an integral component of OpenOffice.org[5], and to Andy Wardley for Template.pm, and its excellent documentation[6].

4. http://writer2latex.sourceforge.net/
5. http://extensions.services.openoffice.org/project/writer2latex
6. http://template-toolkit.org/docs/index.html

5 Afterword - Worked example

5.1 The evaluation report

The Agency has sent me a Microsoft Word file with boxes for my evaluation of the “Activities” and the “Results”, the “Achievements" and the “Methodology", of each “Task’ of each “Work Package” of this project. But their file is just a skeleton. Before I can even start, the pages labelled “Work Package” have got to be copied and pasted n times (for the project’s n work packages).

However, Perl has read the project Report, counted the Tasks and Work Packages, and made me an easy-to-use ASCII file. The first few lines look like this:-

```
!WPNumber                 :: 1
% TASK ACTIVITIES AND RESULTS
!TaskNumber               :: 1
!TaskActivitiesAndResults ::  ...
!TaskNumber               :: 2
!TaskActivitiesAndResults ::  ...

% CONCLUSION 1 OF 2 FOR THIS  WP -  ACHIEVEMENTS
!WPAchievements_etc       :: ...

% CONCLUSION 2 OF 2 FOR THIS  WP -  METHODOLOGY (IES)
% Reminder - choose between Unacceptable Poor Satisfactory Good Excellent
!WPMethodologyies         :: ...
!WPScoreAchievement       :: ...
!WPScoreMethodology       :: ...
```

Now, writing my Evaluation is now as simple as filling in the dots in the ASCII file. That done, we have to make the final report. For this we need a second Perl script which will invoke Perl Template appropriately. Three lines of Perl will do the trick

```
use Template;
my $tt = Template->new({ ABSOLUTE=>1});
$tt->process($template_file, \%hash, $outfile)
  || die "$tt->error";
```

That script fragment first instantiates a variable `$tt`. Then it processes the `$template_file`, using the `%hash`, thus making the LaTeX `$outfile`, all ready to run. The magic, of course, is contained in the `$template_file` and the `%hash`. I discuss these next.

5.2 The template

Lets start with `$template_file`. This began life as the Agency's blank report file, which they made in Microsoft Word. I've explained how I used Open Office to transform that file from Microsoft Word to LaTeX. Then at each point where something has to be filled in, I have put an incantation like this – `[% ...%]`.

For example, the `$template_file` begins so (I've skipped the preamble) ...

```
\begin{document}
  {\centering\sffamily\bfseries A TITLE ... \par}
  \bigskip
    \noindent{\sffamily\bfseries A SUBTITLE
    {[% Project.RefNo %]}
  }\\
  {\sffamily\bfseries Acronym: {[% Project.ShortTitle %]}
    }\\
```

Perl Template replaces each template element – for example `[% Project.ShortTitle %]` – by text from its associative array (discussed below). Perl Template will do this repetitively - i.e. it provides for loops. Here is an example of such a loop (here indexed by the scalar `$a`)

```
\section[OBJECTIVES, TASKS \ldots OF EACH WORK PACKAGE]
  {\sffamily OBJECTIVES, TASKS AND DELIVERABLES OF EACH WORK PACKAGE}
    [% FOREACH a IN WorkPackage.keys.sort -%]
      \subsection{Work Package {[% WorkPackage.$a.WPNumber %]}:
       {[% WorkPackage.$a.WPName %]}
   }
  {\sffamily {[% WorkPackage.$a.WPLeadPartnerNo %]}
```

That fragment was cut and pasted from `$template_file`. Perl Template runs a `FOREACH` loop, filling in the work package number, title, and my assessments, in the correct order.

5.3 The associative array

The text elements are stored in Perl's associative array `%hash`. The first few lines of the `%hash` look something like this

```
%hash = (
 'Project' => {
   'Email' => 'father.christmas@polenord.fi',
   'FaxNo' => '+ 358 0 1234567',
   'Finish_ddmmyyyy' => '01/04/2011',
   'FirstNameLastName' => 'Dr Santa Claus',
  },
 'Objective' => {
  '1' => {
   '1' => {
     'ObjectiveName' => ' Data collection'
    },
   '2' => {
     'ObjectiveName' => ' Data recording into database '
    }
  },
```

All the administrative items, e.g. `[% WorkPackage.$a.WPLeadPartnerNo %]`, come from Perl's parsing of the Report sent in by the project coordinator. The remaining comments and evaluations, e.g. `[% Project.AchievementsAllWPs %]`, come from Perl's parsing of my evaluation report.

5.4 Final comment

The form and the content of the evaluation report have been separated out. The form is a LaTeX document, made by Open Office org from the Agency's report form. The content is contained in a (Perl) associative array. Some comes from Perl's parsing of the coordinators report. The rest of the content comes from Perl's parsing of my evaluation form. The final LaTeX document, uniting content with form, is made by Perl Template.