TEMPLATE FILE PROCESSING

JIM CHEETHAM

The Template Toolkit (TT) provides a metalanguage that can be inserted into otherwise ordinary data files, allowing you to embed data processing instructions.

TT is a collection of Perl modules, and so you will need to have Perl on your system to use TT. Don't worry that you need to understand Perl in order to use TT, though – the template language has been designed to be useable by non-Perl hackers, and you can invoke it simply from the command line.

Scope

TT describes itself as "a fast, flexible, powerful and extensible template processing system". I won't dwell on the speed aspect (TT will save you plenty of time once you're using it) nor the full extensibility (which is achieved primarily through the internal use of Perl).

However, flexibility and power are TT's watchwords. Originally designed for generating dynamic Web content, TT is applicable to a much wider range of tasks. For the purposes of examples in this article, I'll be describing a system for quickly building a set of static Web pages, using the command line tools tpage and ttree.

Installing Template

The current version of Template Toolkit is 2.02, and it is available from the main website, *http://www.template-toolkit.org*.

For those of you used to Perl, it is also available from the CPAN archives, *http://www.cpan.org*. Install with the normal cpan commands:

```
$ Perl -MCPAN -e shell
cpan> install Template
```

The example website

Throughout this article, I will be providing examples from a simple website. Because I'm keeping the examples short, the website might look a little contrived, but I hope you can see the wider applications of TT.

The site will consist of only a few files; initially we will meet only index.html (which is the homepage) about.html which provides some contact details, and info.html, which provides some more information about the site.

To go with these files, we'll use a couple of template files, siteheader.tt2 and sitefooter.tt2. The exact names of all these files is pretty much unimportant, and the extension (.tt2, .html) is doubly unimportant. I just tend to keep using file name extensions like this to help me organise my files while I'm working on them, and they are especially useful if you ever find yourself editing files in a Windows environment.

As the examples build up, more files and templates will be introduced. The example site, and the code used to produce it, can be found on *http://tt.gonzul.net*

The language

The TT language is embedded into your data files, and by default the TT commands are identified by [% and

siteheader.tt2

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html>
[% DEFAULT
       title="TT example site"
       papercol="#ffffff"
       inkcol="#000000"
%]
<head>
<title>[% title %]</title>
</head>
<body bgcolor="[% papercol %]" text="[% inkcol %]">
```

index.html

```
[% PROCESS siteheader.tt2 %]
<h1>TT Example website</h1>
<p>Welcome to the example TT website</p>
<p>Have a look at the other site pages, and don't forget to look at
the source HTML code</p>
<ul>
<li><a href="info.html">Info</a> about the site</li>
<li><a href="about.html">Contact</a> information for the
site</li>
</ul>
[% PROCESS sitefooter.tt2 %]
```

%]. These can, of course, be changed in case they would cause a conflict with your data – TT is flexible, after all. Taking the siteheader.tt2 file as our example:

The file is a fragment of HTML code – specifically, it's the document declaration, header and beginning of the body of an XHTML file. But don't worry about that at the moment, because I'm going to that part of things in a minute.

The <title> and <body> lines are interesting – they show what you will probably recognise as normal HTML lines of code, except that where you would expect to find text (in the case of <title>) or values (<body>) you find a TT code reference to a variable.

Earlier on in the snippet there's a section called DEFAULT, which introduces values for the variables that I'm using below. All the variables look like just plain text – if you want to use real numbers for something (and potentially do some operations on those numbers, like addition or subtraction) you can, trusting the underlying Perl system to Do The Right Thing and automatically transform from text to numeric, and back again, according to context.

When this file is processed by TT, everything it finds between [% and %] will be replaced with the value TT comes up with at the time. So, with the variable title set to "TT example site", the code

```
<title>[% title %]</title>
```

will become

```
<title>TT example code</title>
```

Notice here that the quote marks ("") used to declare the value of title have not been kept, nor have the spaces within the [% title %] section.

Using templates by name

Now, this siteheader.tt2 file isn't very useful on it's own – it won't produce a valid HTML file, for a start. But I can include it at the beginning of every "real" HTML page in my site, by using the PROCESS directive. There are a couple of other variations on this command, called INCLUDE and INSERT, but they don't do quite what I want here. Have a look at the example site's homepage, index.html.

Here I have a simple HTML file, but it doesn't start with <HTML> or even <BODY>, and therefore isn't really a suitable homepage. Instead, it has a TT directive at the beginning, [% PROCESS siteheader.tt2 %]. Similarly, it doesn't end with </body></html> as you might expect, but it does have a TT directive to process the file sitefooter.tt2.

The PROCESS directive allows you to include another template into the current file, and it will keep track of all the variables that you are currently using. This will become clearer in the next example, but for the time being let's just see what happens to our index.html.

I'll use the tpage command to actually process the files. All tpage does is to read in the file you specify, and to run it through a Template instance within Perl, with the results coming out on STDOUT. If you want to see a practical example of how to use Template from within a Perl environment, start by having a look at the internals of tpage – however I'm not going to cover that aspect of Template Toolkit here.

You can see that above and below the actual HTML code from index.html, there appears extra HTML code, that is produced by the siteheader.tt2 and sitefooter.tt2 files. This code has been inserted into the output, and in the case of siteheader.tt2, the variable name references between [% and %] have been substituted for their values. So we now have simple way to make sure that all our files have a consistent header and footer, in just one TT command.

Now, if you actually wanted to look at this file in a Web browser, you'd have to save this output, and put it somewhere sensible, then ask your browser to read that file. But don't worry about that just at the moment, because we haven't met the extremely useful ttree command yet.

tpage

```
$ <span class="pcode">tpage</span> index.html
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>TT example site</title>
</head>
<body bgcolor="#ffffff" text="#000000">
<h1>TT Example website</h1>
<p>Welcome to the example TT website</p>
<p>Have a look at the other site pages, and don't forget to look at
the source HTML code</p>
<ul>
<li><a href="info.html">Info</a> about the site</li>
<li><a href="about.html">Contact</a> information for the
site</li>
</ul>
<div>
<p>Example site copyright &copy; 2001 Jim Cheetham</p>
</div>
</body>
</html>
```

info.html

```
[% PROCESS siteheader.tt2 title="Site Information"%]
<h1>Information about the TT Example website</h1>
<p>The TT example website has been produced to illustrate the use of
<a href="http://www.template-toolkit.org">Template Toolkit</a>
when building static Web sites.</p>
<p>Have a look at the other site pages, and don't forget to look at
the source HTML code</p>
<ul>
<li><a href="index.html">Index</a> page for the site</li>
<li><a href="about.html">Contact</a> information for the
site</li>
</ul>
[% PROCESS sitefooter.tt2 %]
```

So far our example hasn't shown any of TT's more powerful features. PROCESS looks useful enough, but you probably don't want to have all of your pages with the same <title> string, for example. Having multiple siteheader files would defeat the object of using TT in the first place, so how can we easily ask for variations? For the answer to this, have a look at the info.html file.

This file is almost identical to index.html, and you can again see the usefulness of having standard PROCESS instructions to keep our site pages consistent. But for this page, we want to have a different <title>, so in the PROCESS statement where we call for the siteheader.tt2 template, we have included the name and value of the title variable. When this gets processed by tpage, we see the results as the modified listing.

In this case, the <title> declaration in the code now reads "Site Information", instead of the default "TT example site". This flexibility in variable declaration is a big feature for TT. The siteheader.tt2 file does set it's own value for title, but we are able to override it with the PROCESS line in info.html because it is defined within a DEFAULT block, which only sets values for variables if they have not been already specified elsewhere.

Similarly, it would be easy to alter the values of papercol and inkcol on a per- page basis, by including their specifications on the relevant PROCESS lines.

Modified Listing

```
$ <span class="pcode">tpage</span> info.html
?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Site Information</title>
</head>
<body bgcolor="#ffffff" text="#000000">
<h1>Information about the TT Example website</h1>
<p>The TT example website has been produced to illustrate the use of
<a href="http://www.template-toolkit.org">Template Toolkit</a>
when building static Web sites.</p>
<p>Have a look at the other site pages, and don't forget to look at
the source HTML code</p>
<ul>
<li><a href="index.html">Index</a> page for the site</li>
<li><a href="about.html">Contact</a> information for the
site</li>
</ul>
<div>
<p>Example site copyright &copy; 2001 Jim Cheetham</p>
</div>
</body>
</html>
```

menu.html

```
[% PROCESS siteheader.tt2 title="Menu example" %]
<h1>Menu example</h1>
<p>This is an example of a list of values, used twice by the same
template
and presented in two different ways</p>
[% menuitems = [ "first", "second", "third", "fourth", "fifth" ] %]
<table border="1">
<tr><th>Vertical Menu</th><th>Horizontal
Menu</th></tr>
<tr><td>[% PROCESS menu.tt2 dirn="vertical" %]</td>
    <td>[% PROCESS menu.tt2 dirn="horizontal" %]</td></tr>
</table>
[% PROCESS sitefooter.tt2 %]
```

menu.tt2

```
[% DEFAULT
       dirn = "horizontal"
       menuitems = [ "firstitem", "middleitem", "lastitem" ]
%]
[% itemcount = 0 %]
[% FOREACH item = menuitems %]
       [% item %]
       [% itemcount = itemcount + 1 %]
       [% IF itemcount != menuitems.size %]
              [% IF dirn == "horizontal" %]
                     ,
              [% ELSE %]
                     ,<br />
              [% END %]
       [% END %]
[% END %]
```

Decisions, decisions...

So far the templates we've been using have been pretty straight-forward, just setting and using values. TT starts to get more interesting when you encourage your templates to make decisions (based on the values of variables) and produce different output in response.

Let's have a look at a new Web page, menu.html, which uses the template menu.tt2 to present two different variations of the same menu – a little contrived, perhaps, but you will see what I'm getting at.

I'm also going to introduce you to some looping control statements, and list variables. It'll sound easier when you see the examples in the menu.html file:

We've seen most of this before, the two PROCESS directives at the beginning and end are the same as in the other HTML files. However, there are a couple of new things here. The first is the declaration of menuitems as a list of values, in a syntax that Perl people will be familiar with, and the second is the use of a PROCESS directive right in the middle of a table, twice. Each time menu.tt2 is called we are selecting a different value for dirn, the variable that determines how the menu items will be presented. Now for a look at the menu.tt2 template file:

There are a few familiar directives in this file, so let's deal with them first. The DEFAULT block at the beginning allows the template to set values for dirn and menuitems, in case the calling program did not specify them. I'm not so sure how useful it is for a template to provide it's own data, in menuitems. It might be better for the template to check to see if menuitems has been specified, and if not, to output some sort of diagnostic message. It's a matter of taste, I guess.

Then we set the itemcount variable to be 0. This variable will be used to keep track of how far through the list of items we have progressed.

Now we encounter a new directive, FOREACH. This statement sets up a loop construct, which intends to step through each value in menuitems in turn, setting the variable item to whatever the next list item is, each time. The end of the FOREACH block is indicated by the [% END %] statement, and I've used indenting to make it easier for the reader to match up the END statement with the relevant beginning.

The first thing we do inside the loop is to output the current list item value. Then we add one to the counter itemcount. Yes, there are lots of other ways of doing this job, but let's stick to the simple methods.

Now we have a decision to make. If we have reached the last item in the list, we just want to finish. You haven't yet seen what we do if we're in the middle of the list, so it may not be entirely clear why we don't want to do it at the end of the list, but trust me for the moment, and read on.

The decision is made by the [% IF statement. It looks at the conditional, which is the statement "itemcount !=menuitems.size", and decides whether it is true or not. If it is true, in other words, if we are not on the last item in the list, then we can carry on down to the next section of code, otherwise the test fails and we END the IF block.

There is a handy reversal of the IF statement, known as the UNLESS statement. Sometimes it's easier to read your template code in a more natural voice when the test word is the opposite way round. Template Toolkit (and Perl!) tries to be easy to use.

So, we've decided that we're not yet at the end of the list. We'd like to put something between the items in the list, otherwise they'll run together on the final output. For the horizontal list, we'll add just a ",", and for the vertical list, we'll add a comma and a line-break, ",
" (Note that I'm using XHTML statements here, it's good practice and won't break existing browsers).

I'll use a simple IF test on the dirn variable, to see if it is equal to the word "horizontal". If it is, we'll output just a comma, and if it isn't, we'll go for the comma and line-break. Of course, with a test like this we're not being very thorough - if someone had set dirn to a value like "sideways" they'd end up with vertical. You could change the test around to have a different default, or even allow a situation where you could output the list with no delimiters when the dirn is not recognised.

When you get round to running this example, you might be surprised to see lots and lots of extra, blank lines in the final output. This is a side effect of the default behaviour of TT, where it leaves the original file untouched outside of the [% ...%] blocks. This includes the line endings after the %] sections, and can be dealt with - but not with the simple tpage program. Hold your horses and wait for the ttree command, coming up next.

ttree listing

```
$ ttree -d destdir -s sourcedir –ignore
”.tt2$”
ttree 2.03 (Template Toolkit version 2.00)

     Source: ~/ttexamples/tt-website/source/
 Destination: ~/ttexamples/tt-website/example/
Include Path: [ ./Websrc/templates, /usr/local/templates/lib ]
      Ignore: [ \b(CVS|RCS)\b, ^#, .tt2$ ]
        Copy: [ \.png$, \.gif$ ]
      Accept: [ * ]
  + about.html
  + index.html
  + info.html
  + menu.html
  - menu.tt2                      (ignored, matches /.tt2$/)
  - sitefooter.tt2                 (ignored, matches /.tt2$/)
  - siteheader.tt2                 (ignored, matches /.tt2$/)
  - sitemap.tt2                   (ignored, matches /.tt2$/)
```

The ttree command

So far I've been running TT by using tpage on one file at a time, which is fairly awkward and definitely not easy to keep track of. It is time to move up to the ttree program, which offers far more flexibility, and by default will process all of your files correctly.

ttree has a great configuration file, but I unfortunately don't have the space here to explain it – instead, try reading the extensive documentation that comes with Template Toolkit. When ttree first runs, it will try to create a suitable config file, in your home directory by default, and you can go off and edit this to suit yourself.

However, for our example we don't really need anything more complex that the default config file. It is useful to be able to specify the output directory for ttree to be different from the input directory – but don't panic, ttree quite sensibly refuses to let them be the same. So on the ttree command line we'll specify the source and destination directories, and we'll also make sure that it doesn't process the template files we have, by asking it to ignore all files ending in ".tt2". See ttree listing

ttree is your friend. I haven't really been able to do it justice here, beyond the simplest use, but it is an excellent way to help you look after your TT source files and get them built into the right place. It also understands the modification date stamps on your source files, and will only process files that have actually changed, next time you run it. This is best appreciated when running on under-powered workstations, which is pretty much what everyone has.

You might remember from above the comments about tpage allowing what may seem like excessive blank space to appear in your output files. Well, with ttree you can request that TT eats up all that blank space, with a series of options to the command that look like this:

```
$ ttree –pre_chomp –post_chomp –trim
(everything else)
```

Links

The example website, found at
http://tt.gonzul.net
Template Toolkit
http://www.template-toolkit.org
Perl
http://www.Perl.org
CPAN
http://www.cpan.org

Summary

This has been a brief overview of the Template Toolkit's capabilities, introducing basic invocation methods and some simple logic and flow-control directives. With just these commands, however, it is possible to produce some quite complex static websites relatively quickly. Many of TT's more powerful commands are more suited to dynamic work, or to invocation from within a Perl program – which I definitely encourage you to explore in the future! ■