

*Copyright © 2005  
Randal L. Schwartz  
Stonehenge Consulting Services, Inc.  
+1 (503) 777 0095  
<http://www.stonehenge.com/merlyn/>*

*This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.*

# Template Tutorial

---

Randal L. Schwartz  
Stonehenge Consulting Services  
merlyn@stonehenge.com  
Version 1.2 (2/23/05)

# Overview

- What is Template Toolkit?
- The TT language
- Using TT command-line tools
- Using TT from Apache::Template

# What is Template Toolkit?

- Yet Another Templating System
- .. But a good one!
- Actively developed by Andy Wardley
- Driven by needs of etoys.com and Slashcode
- Very active user community

# Key features

- “mini-language” means no learning Perl for those who don’t want or need it
- Compiles to Perl code, cached on disk and in memory
- Very flexible and pluggable
- Works well with mod\_perl
- Many configuration options available
- Can also embed Perl if you prefer that style

# The TT Language

- Control-flow instructions are called “directives”
- Variables map to Perl scalars, hashes, arrays, objects
- TT source can “include” other source
- Use TT language for presentation, Perl for business logic, and SQL for database access

# Simple introduction

- Dear [% name %],

It has come to our attention that your account is in arrears to the sum of [% debt %].

Please settle your account before [% deadline %] or we will be forced to revoke your License to Thrill.

The Management.

# Embedded directives

- Simplest: [% variable %]
  - interpolates a variable
- [% variable = some + calculation %]
  - sets a variable
- Tags can be changed if needed (many styles supported)
- Whitespace within [% .. %] is ignored (mostly)
- Comments are # to end of line (ala Perl)
  - [% a = 3 # first constant  
+ 4 # second constant  
%]

# Chomping whitespace

- By default, newlines remain newlines
- Hello [% a = 3 %]  
World [% a %]
  - Prints “Hello \nWorld 3\n”
- You can absorb whitespace with “-” next to either “%”
- Hello [% a = 3 -%]  
World [% a -%]
  - Prints “Hello World 3”

# Block directives

- FOREACH, WHILE, BLOCK, etc
- Active until the next END
- May be nested
- Output may be captured
  - [% result = FOREACH user = userlist %]  
one user is [% user.name %]  
[% END %]

# Directives alternate syntax

- Directives can be semicolon-separated
  - `{% result = FOREACH user = userlist %}`  
`one user is {% user.name; END %}`
- Often used in place of `.. %}` `{% ..`
- Directives can trail other directives
  - `{% b = b * a FOREACH a = {1, 2, 3} %}`

# The GET directive

- GET foo
- Same as “foo” typically
- Use explicit form when a side-effect calculation should also be interpolated
- CALL is opposite of GET: do the calc, but never insert the value

# The SET directive

- SET var = expression
- Often written “var = expression”
- DEFAULT var = expression
  - set var only if var is false

# Expressions

- Expressions are fairly Perl-like
- ‘single quotes’
- “double quotes with \$variables”
- Concatenation is underscore, not dot
  - this = that \_ other

# The INSERT directive

- `INSERT myfile`
- Include the contents
- Contents are not examined for more `[% .. %]`
- Location is along the `INCLUDE_PATH`

# The INCLUDE directive

- INCLUDE myfile
- Like INSERT, but the contents are TT language
- Might also be a defined block
- Not necessarily a file
- Current variables visible to included template
- Variables set in the included template are not propagated back to the current template
- Extra local variables can be defined
  - [% INCLUDE myfile this = “that” %]

# The PROCESS directive

- Like INCLUDE, but no localization done
- Slightly faster
- More dangerous, since the namespace is shared

# The WRAPPER directive

- `{% WRAPPER foo %}`  
some stuff  
`{% END %}`
- Similar to:  
`{% INCLUDE foo content = "some stuff" %}`
- Great for writing wrappers
- `{% INCLUDE comment_label  
WRAPPER my_button color='blue' %}`

# The BLOCK directive

- Defines named component blocks within current template
- Can be included with INCLUDE, PROCESS, WRAPPER
- `[% BLOCK b %]<b>[%content%]</b>[% END %]`  
`[% WRAPPER b %]my text[% END %]`  
`[% INCLUDE myfile WRAPPER b %]`
- `[% disclaimer = BLOCK %]`  
Portions of tonight's show not affecting the outcome were edited.  
`[% IF sorry %] We're sorry. [% END ; END %]`

# The IF / UNLESS / ELSIF / ELSE directives

- Standard Perl usage
- `{% IF age < 10 %}`  
Hello `{% name %}`, does your mother know you're using her AOL account?  
`{% ELSIF age < 18 %}`  
Sorry, you're not old enough to enter (and too dumb to lie about your age)  
`{% ELSE %}`  
Welcome `{% name %}`.  
`{% END %}`

# The SWITCH / CASE directives

- [% SWITCH myvar %]  
[% CASE value1 %]  
that value  
[% CASE {value2 value3} %]  
either of those  
[% CASE myhash.keys %]  
any of those keys  
[% CASE %] default  
[% END %]

# The FOREACH directive

- `{% FOREACH thing = [foo 'Bar' "$foo Baz" ] %}`  
  `* {% thing %}`  
  `{% END %}`
- Hash shortcut for subentry permitted if variable omitted
  - `{% userlist = [ { id => 'merlyn', name => 'Randal' }`  
  `{ id => 'fred', name => 'Fred Flintstone' } ] %}`  
  `{% FOREACH u = userlist; u.id; " is "; u.name; "\n"; END %}`  
  `{% FOREACH userlist %}`  
    `{% id %} is {% name %}`  
  `{% END %}`
- Nested loops are supported

# Special loop controls

- NEXT and LAST are supported
- The “loop” variable give info on current iterator
  - loop.size = number of iterations total
  - loop.max = size - 1
  - loop.index = current iteration
  - loop.count = index + 1
  - loop.first = true if this is the first time
  - loop.last = true if this is the last time
  - loop.prev = previous item
  - loop.next = next item

# Using special loop controls

- `{% FOREACH i = [ 'foo', 'bar', 'baz' ] %}`  
`{% IF loop.first %}<ul>{% END %}`  
`<li>{% loop.count %} of {% loop.size %}: {% i %}`  
`{% IF loop.last %}</ul>{% END %}`  
`{% END %}`
- `<ul><li>1 of 3: foo`  
`<li>2 of 3: bar`  
`<li>3 of 3: baz</ul>`

# The WHILE directive

- Like Perl
- `{% WHILE total < 100 %} {% total %} {% total += 1 %} {% END %}`
- NEXT and LAST work as in FOREACH
- Warning: 1000 iterations maximum (by default)

# The FILTER directive

- Process a block through a “filter”, return result
- Many standard filters provided
- User defined filters can be created from Perl code
- `{% FILTER html %}`  
Now I can use `<` and `>`  
`{% END %}`
- `{% INCLUDE someblock FILTER html %}`

# Some standard filters

- FILTER format(format) - line by line filtering
  - [% FILTER format('<!-- %0-40s -->') %0]  
some text  
some more text  
[% END %0]
- FILTER upper - uppercase
- FILTER lower - lowercase
- FILTER ucfirst, FILTER lcfirst - as you might expect

# Whitespace filters

- FILTER trim - remove all leading and trailing whitespace
- FILTER collapse - replace all embedded whitespace with single space
- FILTER indent(pad) - indent lines by “pad” spaces
- FILTER truncate(length) - trim to “length” chars, or append dots

# Web-ish filters

- FILTER html - fix up < and > and &
- FILTER html\_entity - fix up latin-1
- FILTER html\_para - turn blank lines into paragraph marks
- FILTER html\_break - turn blank lines into <br>
- FILTER uri
  - %-ifies a string
  - <a href =“ [% filename | uri | html %]”>[% filename | html %]</a>

# Manipulation filters

- FILTER repeat(n) - make “n” copies
- FILTER remove(regex) - remove regex
- FILTER replace(regex, replace) - search for regex, replace with replace

# File filters

- FILTER redirect(filename) - divert output to the file named filename
- FILTER eval - evaluate TT code (normally unneeded)
- FILTER perl - evaluate Perl code (if enabled)
- FILTER stdout - force STDOUT for output
- FILTER stderr - force STDERR for output
- FILTER null - discard output

# The USE directive

- Brings in a “plugin”
- Many pre-defined plugins
- User-defined plugins can be created with Perl
- Defines an object
- Object then gets method calls against it

# The CGI plugin

- [% USE CGI %]
- [% name = CGI.param('param\_name') %]
- [% CGI.start\_form;  
CGI.popup\_menu(  
Name => 'color',  
Values => [ 'Green', 'Brown' ] );  
CGI.submit;  
CGI.end\_form; %]

# The Date plugin

- `{% USE date %}`
- The time is now `{% date.format %}`.
- This file was last modified  
`{%- date.format(template.modtime) %}`

# The Table plugin

- Good for niggly layout tasks
- `{% USE table(list, rows=n, cols=n, overlap=n, pad=0) %}`
- `{% USE table(['a'..'z'], cols=3, pad=0);  
 FOREACH g = table.cols %}  
 [ {% g.first %} - {% g.last %} ( {% g.size %}  
 letters) ]  
{% END %}`
  - [ a - i ( 9 letters ) ]
  - [ j - r ( 9 letters ) ]
  - [ s - z ( 8 letters ) ]

# Other plugins

- Autoformat, Datafile, DBI, Directory, Dumper, File, Filter, Format, GD::\*, HTML, Iterator, Pod, String, URL, Wrap, XML::RSS, XML::Simple, XML::Style, XML::XPath

# The PERL and RAWPERL directives

- Not normally enabled
- Allows direct embedding of Perl code into templates
- Generally a bad idea
- Good for quick-n-dirty playing though
- Eventually, migrate the code into a plugin or filter

# Exception handling

- Permits reasonable exception handling
- Modeled after Java, or recent Perl releases
- Can catch all or selected errors from any TT error
- Can also catch “die” from plugin-thrown Perl code

# The META directive

- Provides meta-data for a template
- Useful in wrapper templates
- Available as “template.thingy” for [% META thingy = ‘this’ %]
- Values are necessarily very simple
- Not a full expression handler

# The TAGS directive

- Changes the [% .. %] markers for remaining included text
- Similar to TAGS and TAG\_STYLE configuration options
- Useful when those characters must be included
- [% TAGS html %]  
<!-- INCLUDE somefile -->

# Variables

- Alphanumerics and underscore, typically lowercase
- Some uppercase-only names are reserved
- Any Perl type (scalar, arrayref, hashref, subroutineref, object)
- Dot notation for accessing items or calling methods
- Trailing parens for calling subroutines
- Keys in hashes that start with underscore or dot are invisible

# Literal values

- Similar to Perl
- Hashes
  - `d = { Fred => 'Flintstone', Barney => 'Rubble' }; d.Fred`
- Arrays
  - `n = [ value1, value2, value3 ]; n.2`
- Dot-dot for ranges: `[ 10 .. 20 ]`
- Can be nested arbitrarily
  - `people = { 'flintstones' => ['fred', 'wilma'], 'rubbls' => ['barney', 'betty'] }`

# Localization

- INCLUDE/WRAPPER creates a new local scope
- PROCESS does not
- Top-level variables defined in local scope are restored on exit
- Secondary variables are not
- Special “global” top-level hash for persistence

# Operations on scalar data

- defined, length, repeat(n), replace(search, replace)
- match(pattern) - match a regex
  - [% name = 'Larry Wall'; matches = name.match('('(\w+) (\w+)') %]
- split(pattern), chunk(size)

# Operations on hash data

- keys, values, each
- defined, exists
- item(key)

# Operations on list data

- first, last - first and last item of list
- size, max - size and size - 1
- reverse, join(string), grep(regex)
- sort, nsort - string sort, numerical sort
- unshift(item), push(item), shift, pop
- unique, merge
- slice(from, to), splice(offset, length, list)

# Using TT from Perl

- use Template;  
my \$template = Template->new(\%CONFIG);  
\$template->process(\$filename, \%VARS);
- %CONFIG is hash of configuration parameters
- %VARS is a hash of predefined global variables
  - %VARS = qw(a 1 b 2);
  - a is [% a %]
  - Prints “ a is 1”

# Configuration parameters

- Not all described here (far too many)
- INCLUDE\_PATH => '/direc/tory:/direc:tory2'
  - Defines where INCLUDE and friends fetch from
- EVAL\_PERL => I
  - Permit [% PERL %] blocks

# Processing configuration parameters

- PRE\_PROCESS, POST\_PROCESS
  - Always PROCESS the given template before/after the requested template
- PROCESS
  - PROCESS this template instead of the current one
  - Current one is available in via [% PROCESS \$template %]
  - META vars of template available as “template.metavarname”

# Caching configuration parameters

- `CACHE_SIZE` - number of templates in memory
- `COMPILE_DIR` - place to cache templates on disk

# Plugin/filter configuration

- PLUGINS - gives prefix for mapping plugin names to package names
- LOAD\_PERL => I
  - Permit “normal” objects as plugins
- FILTERS - specify custom filters

# Using TT command-line tools

- tpage - process a single file
- ttree - process a tree of files, with Makefile-like minimization

# Using tpage

- `$ tpage [ --define var=value .. ] file(s)`
- Great way to test things quickly from command line

# Using ttree

- `$ ttree [options] [files]`
- Reads `.ttreerc` from home dir, or file specified as `-f configfile`
- Processes files recursively from `src` into `dest`
- The `lib` param adds `INCLUDE_PATH` elements
- Other controls: `ignore`, `copy`, `accept`
- Use `ttree --help` for all the gory details

# Using TT from Apache::Template

- Like using Apache::Registry
- URL maps into a top-level template
- Template is recompiled as needed
- Template can use all normal TT language
- Library paths can be established for common templates, and for local plugins
- Pre/Post/Instead-of processing available
- URL can act like CGI, or just a dynamic page
- No need to shove all “CGI” into a separate area

# Apache::Template configuration

- In httpd.conf:  
PerlModule Apache::Template  
TT2PostChomp On  
TT2IncludePath /usr/local/tt2/templates  
TT2CompileDir /var/tt2/cache  
TT2PreProcess config header  
TT2PostProcess footer  
<Location /tt2>  
  SetHandler perl-script  
  PerlHandler Apache::Template  
</Location>

# Web Widget libraries

- HTML library
  - Provides basic HTML support
- Splash! Library
  - Provides nice colorful widgets
  - Built on the HTML libraries
  - Comes in “themes”

# Powerful higher layers

- OpenInteract - Appserver based on TT
  - Provides common widgets for things like “login”
  - Provides abstraction of an “application” that can be independently upgraded
- Slashcode - Community builder
  - DBI-based templates
  - Login, user profiles
- Bricolage - CMS
  - Can use TT to “burn” content for presentation

# Questions?

