Templating System

Why Templating?

If you've ever built a large website with static HTML pages, it should become rather obvious why templating is needed. To illustrate the reasoning behind this system, I'll give a very simple situation, probably one that you're a little too familiar with (hence why you found this project):

> *You've been contracted to build a static website (no fancy scripting involved) with 10 different pages. Each page needs to have the same look and feel, with the same logo at the top, the same menu beneath that, and the same footer with copyright information and such at the bottom. Between that menu and footer, each page needs to have a separate chunk of content, which are provided (in a perfect world) by somebody else.*
>
> *Jumping onto your workstation, you whip out your favorite graphical HTML editor, and quickly create the first version of the front page, which all other pages are based on. After some tweaking, it's approved. You're off to build the other 9 pages. Maybe the first one or two are done by hand, but soon it boils down to a marathon of copies and pastes. When it's all done, the site goes onto the internet, and everybody is happy.*
>
> *A week later (if you're lucky), that same guy calls, and asks you to update the menu on the front page. While making the rather simple change, you realize that this same change must also be made to the other… 20 pages? That's right, somebody else made a bunch of pages, with a similar layout, and a similar menu. Hours later, after more tedious copy+pasting of HTML, your employer asks for a new layout, with a brand new menu, logo, and footer.*

Now imagine that same scenario, with the exception that the website is completely templated: changing the menu requires modifying one file (if it's setup properly), as does the header and footer. What used to take hours of copying, pasting, and reloading the page, now takes just a few minutes. New pages are created by nothing more than creating a single content file for each page. And somehow, magically, that copyright date on the footer always shows the current year. And in a year, when the website gets a massive redesign, and 300 more links are added, you don't have to worry about shaving off those few shreds of hair that are left from the tedium of modifying hundreds of pages, just to make them look the same.

Definitions and Syntax:

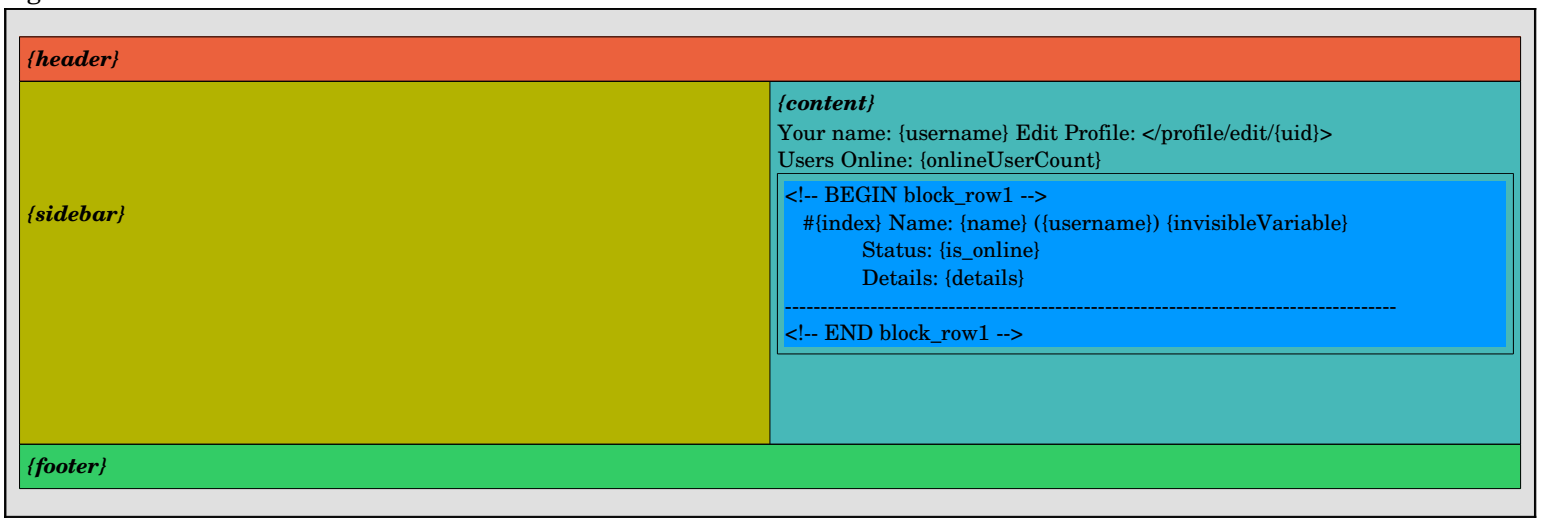
Term	Syntax	Definition
Template file	(varName).(section).tmpl	File that contains information for a specific part of a web page.
Template var	{variable}	a placeholder within a template file or HTML document to describe where the contents of another template or template var will be placed.
Template Section	(special)	This is a special indicator of what type of template a given file is. Presently, there are only three types: system, content, and shared.

Template Sections:

1. Content
   - generic template, which will be loaded only for the URL associated with it's location in the “template” directory.
2. Shared
   - loaded for every URL that at the same level, or deeper, than it's location.
3. System
   - special templates, used by the the templating system, or cs-content, for various reasons.
   - creating new system templates, or modifying existing ones, should be done carefully, and only by a qualified professional. ;)

Basic Layout (/templates/main.shared.tmpl)

Figure 1



Explanation of layout:

There are four main sections (denoted by "*{section}*"):

1. "header": which normally designates the top part of the page
2. "sidebar": contains all menu information
3. "footer": the bottom portion of the page
4. "content": this is normally the only part of the page that constantly changes, depending upon the URL

Each this "main" layout (main is actually a template var that may also be overwritten—be careful) basically explains the basic layout of the page. By modifying this template, one can easily move large pieces of the layout very simply. That is the sole purpose for this template.

Each successive template file then contains smaller bits of the whole: a special "content" template is made for each new page, and any redundant data is stored in little "shared" templates.

Within the "content" section, notice the part that says "<!-- BEGIN block_row1 -->": this is a block row. This is a single row that will be repeated multiple times to show results of a search (think of it as one of the lines from a search on Google). The code can rip this part out into a template var (called "block_row1"), parse the data within it (using a special parser called "mini_parser()"), and append each parsed row into a single variable until all rows have been handled.

Sample code for parsing data into this row:

Figure 2

```
1   <?php
2           $myResults = $searchObject->get_results();
3           $myRowTemplate = $page->set_block_row('content', 'block_row1');
4           $allParsedRows = “”;
5           foreach($myResults as $index=>$data) {
7                   $replacementsArray = $data;
8                   $replacementsArray['index'] = $index;
9                   $allParsedRows = $page->mini_parser($myRowTemplate, $replacementsArray);
10          }
11
    ?>
```

First (line 2, figure 2), retrieve results from the search (“`$searchObject->get_results()`”): this is expected to return an array, with each index representing a single item, and the sub-array of that index containing data for the row. An example result set (for ease in translation, the example declares the value of “`$myResults`”; the code above pulls that data dynamically):

Figure 3

```
1   <?php
2           $myResults = array(
3                   595 => array(
4                           'name'          => 'Johnny Cash',
5                           'username'      => 'jcash',
6                           'details'       => 'no details for you!',
7                           'is_online'     => f
8                   ),
9                   678 => array(
10                          'name'          => 'Dan Falconer',
11                          'username'      => 'crazedsanity',
12                          'details'       => 'Go visit http://www.crazedsanity.com',
13                          'is_online'     => t
14                  ),
15                  777 => array(
16                          'name'          => 'You',
17                          'username'      => 'test',
18                          'details'       => 'Get me a username and password',
19                          'is_online'     => f
20                  )
21          );
22  ?>
```

Back to figure 2: in line 3, the code calls “set_block_row()”, with the arguments “content” and “block_row1”. The first argument (“content”) indicates what template our block row resides, while the second argument (“block_row1”) gives it the name to search for. In figure 1, within the greenish-blue section labeled “{content}”, you'll see exactly how that block row is defined: much like a standard HTML tag, it has a beginning and an end.

On line 4 (figure 2), it defines a template for all the parsed rows to fit into (“`$allParsedRows`”): this is unnecessary, but a good practice. This ensures there is no data in this variable already.

On line 5 (figure 2), a "`foreach`" loop begins: this will go through each row in the result set (see figure 3 for an example), parsing the template for it each time. Lines 7-9 are the heart of the code, handling each row in the result set ("`$myResults`") and making a chunk of HTML out of it. Essentially, the index of each item is used to determine the name of the template var it should replace, and it's associated value is then used to fill in that section. The parsed data should look similar to what is in figure 4.

With each successive loop, the row is re-parsed & appended to the existing data, creating multiple parsed rows.	#595 Name: Johnny Cash (jcash) Status: f Details: no details for you! --- #678 Name: Dan Falconer (crazedsanity) Status: t Details: Go visit http://www.crazedsanity.com --- #777 Name: You (test) Status: f Details: Get me a username and password ---

Notice in the original row template in figure 1 there was a template var, "invisibleVariable", that seems to be missing from the output. The reason it is missing is because once these rows are parsed and CS-Content finishes parsing the entire page, any unfilled template var will be stripped as if it were never there.

Template Inheritance:

CS-Content automatically loads templates based on the URL.

MAIN SECTION:

For the main section, i.e. "/content" (which is synonymous with "/" in the default setup, like "http://www.yoursite.com/"), it requires a template in a directory of that name beneath /templates, with a file called "index.content.tmpl"... i.e. /template/content/index.content.tmpl.

SUB SECTIONS:

For any subsection to be valid, i.e. "/content/members", it must have an associated template, i.e. "/templates/content/members.content.tmpl". If a subdirectory with an index.content.tmpl file exists, it will be used instead of the file in the sub directory (i.e. "/templates/content/members/index.content.tmpl").

SUB SECTION TEMPLATE INHERITANCE:

All pages load the base set of "shared" templates, which are in the form "<section>.shared.tmpl" in the root of the templates directory.

Shared files within each directory, in the form "<section>.shared.tmpl", will be loaded for ANY subsection. For any subsection, it inherits a previous section's templates in the following manner (any "content" templates are ignored for inheritance, as they're required for page load).

```
/content                    ---> /templates/content/index.*.tmpl
/content/members   |        |--> /templates/content/index.*.tmpl
                            `--> /templates/content/members.*.tmpl

/content/members/test       |--> /templates/content/index.*.tmpl
                            |--> /templates/content/members.*.tmpl
                            |--> /templates/content/members/index.*.tmpl
                            `--> /templates/content/members/test.*.tmpl
```

AUTOMATIC INCLUDES:

Much in the same way templates are included, so are scripts, from the /includes directory, though the logic is decidedly simpler: all scripts must have the extension of ".inc", and must have either the section's name as the first part of the filename, or "shared". Shared scripts will be loaded for ALL subsections.

INCLUDES INHERITANCE:

The template inheritance scheme is as laid-out below. The content system will go as far into the includes directory as it can for the given section, regardless of if any intermediate files are missing.

```
/content                    |--> /includes/shared.inc
                            `--> /includes/content.inc

/content/members            |--> /includes/shared.inc
                            |--> /includes/content.inc
                            |--> /includes/content/shared.inc
                            `--> /includes/content/members.inc

/content/members/test       |--> /includes/shared.inc
                            |--> /includes/content.inc
                            |--> /includes/content/shared.inc
                            |--> /includes/content/members.inc
                            |--> /includes/content/members/shared.inc
                            |--> /includes/content/members/test.inc
```