PHP Fast Template

By Ian Felton

Table of Contents

Why separate presentation from logic?

First, what is the difference between presentation and logic. Its the difference between dealing with the appearance of an apple and dealing with its genetic code. With web applications, presentation includes HTML tags, basic Javascript such as rollover effects, FLASH and anything else in that vein. Logic includes all of the software written in Python, Perl, PHP, or the p−p−p−particular language of choice which best solves the problem at hand.
So when writing web applications, why separate the two?

The simple answer to the question of why presentation should be kept separate from logic is, "It keeps things simple". If presentation wasn't separated from logic around your house, you'd have to be an electrician to replace light−switch covers. The same goes for large−scale web applications. Graphic designers shouldn't need to be software engineers in order to update the fonts in a web page. Separating logic from presentation makes that possible.

There are many ways to accomplish this, but in this article we are going to focus on one method using PHP and an external class called FastTemplate. This class can be downloaded from:
http://www.thewebmasters.net/php/

Reminder: Sometimes separating presentation from logic is overkill. Determine the scope of your project.

Placing FastTemplate Variables in HTML Templates

It's very simple to include data from an application's business logic into HTML templates. HTML templates are files consisting of HTML with placeholders for dynamic data.

Whenever dynamic data needs a place holder in the HTML template, place the variable name between curly braces.

Ex:

```
<HTML>
<BODY>
{HOUSE}
<P>
This is a house.
</P>
</BODY>
</HTML>
```

The variables can be anything that the FastTemplate class can parse: alphanumeric symbols and the underscore, or {(A−Z0−9_)+} for you reg ex lovers. Always keep these variables descriptive of the data that will fill it. Remember, graphic designers will ideally be editing these templates, so they need to know exactly what is going there. Creating charts that index which templates contain specific variables is useful if the need ever arises to change the names of any of them.

Including the Fast Template Module in PHP Code

So how does the data get from code into the HTML templates? First, the FastTemplate class file must be included in the PHP code. Accomplish this the same as including any external class file. Here is the code to include the FastTemplate class file in a PHP script:

```
<?php
include "class.FastTemplate.php3";
```

Reminder: Be certain to place the path to the FastTemplate class file into the PHP.ini file's INCLUDE_PATH variable.

Assigning templates to objects

Once the class file has been included, HTML templates can be assigned to an array for use throughout the script.

First, a new template object must be instantiated like this:

```
$tpl = new FastTemplate("../templates");
```

The path in quotes points to the location of the HTML templates being used. This will vary depending upon personal preferences.

Once the instance exists, external HTML templates are assigned to variables in the array like this:

```
$tpl->define(array(
"index_tpl" => "some_html_template.tpl"
));
```

In the above code, "index_tpl" is arbitrary. In fact many solutions involve assigning many templates to variables within one instance.

An example of multiple templates declared for use in a script:

```
$tpl->define(array(
"index_tpl" => "some_html_template.tpl",
"sub_idx_tpl" => "another_html_template.tpl",
"last_idx_tpl" => "last_html_template.tpl"
));
```

Assigning variables to FastTemplate objects

Within the PHP script, variables will eventually need assigned to the FastTemplate instance. Here is the syntax for doing this:

```
$tpl->assign("HOUSE", $House);
```

Now the FastTemplate object contains the data found in the variable, "$House"

Parsing Templates
Parsing the FastTemplate instance remains the final bit of processing that takes place in order to see results. These lines accomplish that task.

```
$tpl->parse(MAIN, "index_tpl");
$tpl->FastPrint();
```

"MAIN" in the above code represents the top level HTML template. Sometimes many HTML templates are parsed and each one will be represented with a distinct name. "MAIN" refers to the parent HTML template. The final command calls the FastPrint function which delivers the final product, a beautiful dynamic HTML page, to the computer screen.

MySQL Data parsed through FastTemplates

Using the above information as a guide, look at this code that works on data taken from a MySQL database and parses it through HTML templates.

```
<?php
//getData.php
include "class.FastTemplate.php3";

//instantiate a new FastTemplate instance
$tpl = new FastTemplate("../templates");

//associate an HTML template to a variable
$tpl->define(array(
"toplevel" => "phone_numbers.tpl"
));

//Connect to database
mysql_connect (localhost, root, passwd);

//select database to use
mysql_select_db (testDB);

//select data from MySQL database for example code
$GetData = mysql_query ("select Phone from Business where Name
= '$Name'");

//Get array results
$GetDataArray = mysql_fetch_array($GetData);

//Associate contents of hash array with variable
$Phone = $GetDataArray["Phone"];

//assign $Phone to FastTemplate instance
$tpl->assign("PHONE", $Phone);

//pass the data to the HTML template
$tpl->parse(MAIN, "toplevel");
$tpl->FastPrint();

?>
```

From here, the phone number data taken from the database will be passed to the HTML template, "phone_numbers.tpl". Here are the contents of phone_numbers.tpl :

```
<HTML>
<BODY>
<P>
Here is the phone number you wanted:
{PHONE}
</P>
</BODY>
</HTML>
```

On the web page it will look like this:

```
Here is the phone number you wanted:

555-5555
```

Multiple templates: Rows of data into a table

Perhaps the data needed for the web page is a table of records from a database. The best way to present it would be through a table. With FastTemplate this can be accomplished cleanly. Look at this variation on the previous example to see how its done.

```
<?php
//getData.php
include "class.FastTemplate.php3";

//instantiate a new FastTemplate instance
$tpl = new FastTemplate("../templates");

//associate three HTML templates to a variables
//toplevel is the parent document, table is parent to the rows
template,
//rows contains the data
$tpl->define(array(
"toplevel" => "phone_numbers.tpl"
"table" => "table.tpl",
"rows" => "rows.tpl"
));

//Connect to database
mysql_connect (localhost, root, passwd);

//select database to use
mysql_select_db (testDB);

//select entire set of records from MySQL database for example
code
$GetData = mysql_query ("select Phone from Business");

//Get array results
if ($GetDataArray = mysql_fetch_array($GetData)) {
do {
//Associate contents of hash array with variable
$Phone = $GetDataArray["Phone"];

//assign $Phone to FastTemplate instance
$tpl->assign("PHONE", $Phone);

//concatenate each row of data as the do loop cycles through
the array of records
$tpl->parse(ROWS, ".rows");
```

```
}while ($GetDataArray = mysql_fetch_array($GetData));

//when all the records have been obtained and parsed through
the ROWS template,
//parse the entire table
$tpl->parse(PHONERECORDS, "table");

//pass the data to the HTML template
$tpl->parse(MAIN, "toplevel");
$tpl->FastPrint();
}

//if no records found, print error message
else {
print("Error obtaining data.");
}

?>
```

In this example, the data for PHONE will be parsed through the ROWS (rows.tpl) template which looks like this:

```
<TR>
<TD>
{PHONE}
</TD>
</TR>
```

Each row will be connected to the end of the previous one. Once they have all been parsed, the entire set will be parsed as whole and placed in the TABLE (table.tpl) template, which looks like this:

```
<TABLE>
{ROWS}
</TABLE>
```

Finally, the entire table of records will parsed through the parent template which looks like this:

```
<HTML>
<BODY>
<P>
Record of phone numbers:
{PHONERECORDS}
</P>
</BODY>
</HTML>
```

Now the entire set of data has been parsed and displayed to the screen cleanly formatted and completely separate from logic. At this point, designers can edit the template files for different fonts, images, text, etc. and leave the code alone.

Summary

Many times, when working with smaller projects, taking the time to separate the presentation from logic is like taking the time to ask guests to leave the dinner table each time a new dish is brought from the kitchen to hide where its coming from. Some programmers have said if lines of code are greater than 1000, go ahead and separate the two. There's no question that projects that are going to involve 100% database stored content can benefit from creating HTML templates and parsing data through them with PHP and the FastTemplate class. Remember that there are many tools for separating presentation and logic and the best one for the job is the one that's desired. Make certain you have the right one before the project gets too deep.

Appendix / Notes
When installing PHP4 and FastTemplate, it was necessary to edit a line of code in the FastTemplate class in order for it to work properly.

Within the parse_template function of the class this line was edited:

```
$template = ereg_replace("\{$key\}","$val","$template");
```

The curly brackets had not been escaped in the code so the script ignored any variables in the HTML templates.