Simple Template System in PHP

Manuel A. Perez-Quinones, Computer Science, VT, 2013
Updated:

- September 22, 2013
- December 27, 2014
- January 28, 2015

This is a very simple template language implemented in PHP to be used for small projects and (initially designed for) classroom instruction. The language supports variables, conditionals, repetitions, calls to functions, and several data manipulations directives.

Variables

Variables can be placed anywhere in the HTML by using the name of a variable and encapsulating it in curly braces. `{name}` for example, will be replaced by the value of the variable. Variable names can be any sequence of a-z, A-Z and 0–9. It cannot contain spaces nor any other special character.

Passing Variables

Variables are passed in an associate array. The index (key) is the name of the variable and the value associated with that key will be used in the substitution. A simple example follows:

Consider an HTML file, `ex1.tmpl`, that includes the following:

```
<!-- stored in ex1.tmpl -->
<html>
<body>
<h1>Welcome</h1>
<p>My name is {name} and this is home page.</p>
</body>
</html>
```

To use this template, we prepare an array with the variables to be passed to the template. Then we call the `gen_template` routine as shown below.

```
require_once("vendor/autoload.php");          // <-- use composer
```

```
// set the variables
$symbols = array('name' => "Joe");

// generate the template
$page = gen_template("ex1.tmpl", $symbols);
echo $page
```

Directives

In addition to variables, the template system supports a group of directives. Directives control how the text is generated by the template system. The directives are embedded in `<% directive %>`. Possible directives are described below.

If directive

The `if` directive allows a condition to be expressed. The condition is a variable passed to the system as shown above. If that variable evaluates to `true`, then the text for the “then-block” will be included in the output. If the variable evaluates to `false`, then the “else-block” will be included in the output. Note that the else block is optional.

```
<% if {condition} %>
    html
<% else %>
    more html
<% end %>
```

Unless directive

The `unless` directive is similar to if but evaluates the condition with a negation. If the condition evaluates to `false`, then the text for the “then-block” will be included in the output. If the variable evaluates to `true`, then the “else-block” will be included in the output. Note that just like in the if statement, the else block is optional.

```
<% unless {condition} %>
    html
<% else %>
    more html
<% end %>
```

Repeat directive

The `repeat` directive, as with the `if`, uses a variable to control the text generation. In this case, however, the variable is an array. The body of the repeat will be included in the ouptput once for each element of the array.

```
<% repeat {collection} %>
    html
<% end %>
```

For example, consider the following representation showing a person's name, and his phone numbers.

```
$symbols = array('name' => "Joe",
    'phones' => array(
        array('type' => "home", 'number' => "555-1234"),
        array('type' => "cellular", 'number' => "555-2345")
        ));
```

At the top level of your variables, you want to have the symbol used in the repeat line. In this example it is `{phones}`, as shown below.

```
$symbols = array('name' => "Joe",
    'phones' => array( .. ));
```

The content of the array (`phone`), should be arrays themselves with definition for symbols used in the repeated part of the repeat statement. In this example, each element of the `{phones}` collection should have `{type}` and `{number}`. Completing the example, we have:

```
$symbols = array('name' => "Joe",
    'phones' => array(
        array('type' => "home", 'number' => "555-1234"),
        array('type' => "cellular", 'number' => "555-2345")
        ));
```

The corresponding template shows how to use the repeat directive:

```
<html>
<body>
<h1>Welcome</h1>
<p>My name is {name} and this is home page.</p>
<ul>
<% repeat {phones} %>
    <li>my {type} number is {number}</li>
<% end %>
</ul>
</body>
</html>
```

The repeat statement also supports additional variables that can be used in the loop body.

- {loopfirst} evaluates to `true` in the first iteration of the loop. It is `false` the rest of the time. It can be used to generate output on the first time through the loop.

- {looplast} is similar to ‘loopfirst’ but evaluates to `true` only on the last iteration through the loop.
- {loophasmore} evaluates to `true` on all iterations through the loop except the last one. Logically same as “not looplast”.
- {loopcount} contains the index of the loop counter.
- {loopodd} evaluates to `true` when ‘loopcount’ is an odd number.
- {loopeven} evaluates to `true` when ‘loopcount’ is an even number.

With these, we could do a more complex text generation using the previous example:

```
<% repeat {phones} %>
    <% if {loopfirst} %>
        <ul>
    <% end %>
    <% if {loopodd} %>
        <li style="background-color:gray;">
    <% else %>
        <li>
    <% end %>
    my {type} number is {number}</li>
    <% if {looplast} %>
        </ul>
    <% end %>
<% end %>
```

Include directive

The `include` directive works like other include statements in programming languages. The `<% include filename %>` will be replaced with the contents of the ‘filename’ after it is expanded with the current set of variables. Note that there are no quotes around the file name. For now, the file name must not include a space. This will be fixed in a future version.

```
<% inlcude ex1.tmpl %>
```

For now, the include directives do not work when combined with the data directive. That is, you cannot include a file with a data directive. This will be fixed in a future update (examples ex8b and ex8c do not work correctly).

Layout directive

The `layout` directive turns the template system inside out. Instead of having a template that includes other files, the layout directive allows the definition of the content of a page that is included in a predefined layout. This command allows the bulk of the layout (e.g., outer layout, headers, footers, etc.) to be defined once and to be generated from the “content” of the page. This makes it easier to change the layout by simply including a different layout.

```
<% layout ex1.tmpl %>
```

More documentation and examples are needed for this feature. It will be completed later.

Call directive

The `call` directive allows calling a PHP function from the template expansion. The function must be defined at the time of the expansion and it can be a PHP function or a user defined function. The associative array with variables passed to the template expansion are passed back to the function being called. The example below shows how this works. The return value from the function will be incorporated directly in place of where the call took place.

The example below has the header section of the HTML being generated by the PHP code instead of the template file. The template simply places a call to the `pageheader` function.

```
<!-- ex5.tmpl -->
<html>
<% call pageheader %>
<body>
<h1>Phones</h1>
<dl>
<dt>Name: {name}</dt>
<dd><% repeat {phones} %>
    {type} number is {number}<br>
<% end %></dd>
</dl>
</body>
</html>
```

The PHP code that uses this template is below. Note that the `pageheader` function is defined to take one argument, the parameters passed to the `gen_template` function. In this example, the `pageheader` generates the HTML using a variable that came from the `gen_template` call ('pagetitle').

```
require_once("vendor/autoload.php");        // <-- use composer

function pageheader($params)
{
    return "<head><title>{$params['pagetitle']}</title></head>\n";
}

$symbols = array('name' => "Joe",
    'phones' => array(
        array('type' => "home", 'number' => "555-1234"),
        array('type' => "cellular", 'number' => "555-2345")
        ),
    'pagetitle' => "Hello there");
echo gen_template("ex5.tmpl", $symbols);
```

Data directive

The `data` directive allows for data to be included in the template file directly. This might make it easy to define values that are used at multiple locations, sort of like constants in a programming language. Currently the data directive supports three formats: csv, json, and xml. We plan to add yaml in the future.

```
<% data {json | xml | csv} %>
_data goes here_
<% end %>
```

What follows is a simple example, just to demonstrate how the data directive works.

JSON

```
<!-- ex7.tmpl -->
<% data json %>
{
    "title":"Ex7",
    "name":"manuel",
    "looping": [
        {
            "name":"var1",
            "value":10
        },
        {
            "name":"var2",
            "value":20
        }
    ]
}
<% end %>

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>{title}</title>
    <meta name="author" content="MAPQ">
  </head>
  <body>
    <h1>{title}</h1>
    <p>My name is {last}, {name}.</p>
   <ul>
        <% repeat {looping} %>
         <li>{name} = {value}</li>
        <% end %>
   </ul>
  </body>
</html>
```

This example includes data stored in json format in the template file itself. Nevertheless, the file must be generated from PHP. The code below shows the PHP code for this example. Worth noting is the variable {last} which is used in the template but is not

included in the json data.

```
require_once("vendor/autoload.php");        // <-- use composer
$symbols = array('last'=>"Perez");
echo gen_template("ex7.tmpl", $symbols);
```

This shows that the symbols passed from the PHP code are extended with the data encountered in the data directive. As a result all the symbols are available for the template.

Similar examples are included below in CSV and XML formats.

CSV

```
<!-- ex8.tmpl -->
<% data csv %>
id,person,score
joe,Joe Smith,100
mary,Mary James, 101
<% end %>

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Grades</title>
    <meta name="author" content="MAPQ">
  </head>
  <body>
    <h1>Grades</h1>
    <p>Professor {last}.</p>
   <table>
        <% repeat {csv} %>
            <% if {loopfirst} %>
            <tr><th>PID</th><th>Person</th><th>Pts</th></tr>
            <% end %>
         <tr><td>{id}</td><td>{person}</td><td>{score}</td></tr>
        <% end %>
   </table>
  </body>
</html>
```

The PHP code for this example:

```
require_once("vendor/autoload.php");        // <-- use composer
$symbols = array('last' => "Perez");
echo gen_template("ex8.tmpl", $symbols);
```

XML

```
<!-- ex11.tmpl -->
<% data xml %>
<xml>
```

```
    <name>manuel</name>
    <last>perez</last>
    <looping>
        <name>var1</name>
        <value>10</value>
    </looping>
    <looping>
        <name>var2</name>
        <value>20</value>
    </looping>
</xml>
<% end %>

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>{title}</title>
    <meta name="author" content="MAPQ">
  </head>
  <body>
    <h1>{title}</h1>
    <p>My name is {last}, {name}.</p>
   <ul>
        <% repeat {looping} %>
         <li>{name} = {value}</li>
        <% end %>
   </ul>
  </body>
</html>
```

The PHP code for this example is:

```
require_once("vendor/autoload.php");        // <-- use composer
$symbols = array('last' => "Perez");
echo gen_template("ex11.tmpl", $symbols);
```

Reference

gen_template_from_json($tmpl_file, $jsonfile)

Generates a page using the template stored in `$tmpl_file` and data stored in JSON format in the file indicated by the second argument, `$jsonfile`.
Returns false if either one of the files doesn't exist.

gen_template_from_csv($tmpl_file, $csvfile)

Generates a page using the template stored in `$tmpl_file` and data stored in csv format in the file indicated by the second argument, `$csvfile`. The first line of the csv file will be used for variable names for the corresponding columns.
Returns false if either one of the files doesn't exist.

gen_template_from_xml($tmpl_file, $xmlfile)

Generates a page using the template stored in `$tmpl_file` and data stored in XML format in the file indicated by the second argument, `$xmlfile`.
Returns false if either one of the files doesn't exist.

gen_template($tmpl_file, $variables)

Generates a page using the template stored in the file `$tmpl_file` and the variables stored (as an associative array) in the second argument.
Returns false if the template file doesn't exist.

Pending items and other ideas

- better error checks and reporting of errors through exception throwing
- add special case to repeat to support a csv array with the first row having the field names and the individual rows not being in an associative array.
  <% repeat {} %>
  <% end %>
- expand the <% include {file} %>
  - as a way to read in another file, recursive call in parsing the file
  - if file is tmpl - parse and incorporate the resulting tokens in parsed stream
  - if file is html - drop the html as a single text entity
  - if file is md - then process markdown and drop as a single text entity
  - if file is csv - then same as data but external file
  - if file is xml - then same as data but external file
  - a loop of includes will kill the system - should we avoid them?
- add SQL by … {service:user:password}
  <% data {mysql:user:password} %>
  SELECT * FROM X WHERE `name`='who'
  <% end %>
- add <% usedata {file} %>
  - reads file, determines type, and generates output

- create new function that allows both file and variables, for example:

```
gen_template("template", "inputfile", $variables);
```

- integrate with Apache via url rewrite, dispatch supports different url syntax
  /template
  /template/data
- small dispatch in .htaccess does all the magic, add “root” as variable
- how best to support markdown? Either as part of data or as separate file

```
<% markdown filename %>
```

the end