O'REILLY
ONLamp.com
LAMP: THE OPEN SOURCE WEB PLATFORM

Published on **ONLamp.com** (http://www.onlamp.com/)
http://www.onlamp.com/pub/a/php/2002/09/05/smarty.html
See this if you're having trouble printing code examples

Introducing Smarty: A PHP Template Engine

by Joao Prado Maia
09/05/2002

Most PHP developers go through scores of changes in regards to their development expertise. They usually start by creating simple scripts to add dynamic features to their Web sites, then go on to add new features. This process leads to more complexity with the mix of PHP and HTML. It is quite common to see PHP scripts that include other files, with the ultimate objective of reusing code and HTML widgets. One that includes file outputs--the standard header of the site; another one the standard table, and so on.

The Reasoning Behind Templates

While that process is certainly valid and useful, most developers agree that the separation of business logic and layout logic makes the code a lot easier to understand and maintain. This is the reason behind templates, to separate business logic from layout.

What Smarty Has to Offer

Smarty is a somewhat new development in the PHP world, and it brings several new and unique features. One of these unique features is that Smarty 'compiles' the parsed templates into PHP scripts, and then reuses the compiled template when appropriate. Obviously, this brings a huge performance improvement over other template solutions, as the main PHP script doesn't need to parse and output the same template on every request.

Smarty also has support for plug-in modules, allowing developers to create their own set of special functions and have Smarty recognize them. And it has built-in caching support, special constructs that can be used on templates to control the format of the layout, and much more.

Most importantly, Smarty gives developers tools that let them separate the business-logic code from the layout-formatting code. And Smarty goes one step further by allowing developers to put control-flow structures in the template source. This might sound a bad idea, since it would imply business-like logic

in the template, but it is actually quite useful--you can tell Smarty to use a specific color on table X on the template itself, instead of having PHP code do this work. After all, this *is* template-related information.

One quick template example might be of value after this quick introduction:

```
{include file="header.tpl.html"}

Welcome to my Web site!

IP Address: {$smarty.server.REMOTE_ADDR}

{include file="footer.tpl.html"}
```

Installing Smarty

Smarty is quite simple to install, and I'll give step-by-step instructions on how to do just that. First, you must download the distribution.

After downloading and extracting the files from the tarball, copy the resulting Smarty directory to some place inside your `include_path`. A good option is to copy this directory in the PEAR library directory (see my previous articles about PEAR). In Unix environments, it will usually be:

```
$ cp -R Smarty /usr/local/lib/php/
```

In Windows computers, you will need to copy the `Smarty` directory to '`C:\php\pear`'.

Organizing Your Application to Use Smarty

There are several ways to organize your application to make use of Smarty, but I'll document here what I always do to structure my Web sites and applications.

My directory structure is as follows:

```
docs
  index.php
  configs (directory)
  templates (directory)
  templates_c (directory)
```

Make sure to give the Apache user write access to the `templates_c` subdirectory. This is the directory that Smarty uses to store the compiled templates. You can do this by using the following commands in a Unix system:

```
$ mkdir templates_c
$ chown nobody:nobody templates_c
$ chmod 700 templates_c
```

Please note that I'm assuming that the Apache user is `nobody`, so you should change the line above accordingly.

Using Smarty

Now that you've installed Smarty on your server, let's get to the basics on using Smarty on your templates. Below is a simple PHP script that manipulates a template and assigns a template variable to a value:

```
<?php
require 'Smarty.class.php';
$smarty = new Smarty;

$smarty->assign("full_name","Tim O'Reilly");

$smarty->display('index.tpl');
?>
```

Before doing anything, the script includes the Smarty class, which is included in `Smarty.class.php`. An object is created and the template variable (sometimes referred to as a template placeholder) named `$full_name` will hold the value "Tim O'Reilly."

Whenever you create your templates, the placeholders will be placed as `{$variable_name}`, but the placeholder delimiter can also be changed. This way you can use `%$variable_name%`, or anything you want.

In any case, the accompanying template file for this script is below:

```
<html>
<body>

O'Reilly & Associates Founder is {$full_name}

</body>
</html>
```

As explained above, `{$full_name}` will be substituted by the string "Tim O'Reilly" and outputted to the browser. Smarty also compiles the template into a PHP script, so the next time this script is requested, the template will not be parsed again (unless it changes) and the compiled PHP script will be used instead.

Control Flow with Smarty

Smarty also provides control-flow tags for developers:

```
{section name="i" loop=$list}
  <b>Iteration {$list[i]}</b><br>
{sectionelse}
  <b>Array $list has no entries</b>
{/section}
```

The `{section}` tag above provides a quick and simple way to loop through a list of values. All you need to do is to assign a variable to the `$list` variable and the `{section}` tag will loop through it. Like so:

```
<?php
require 'Smarty.class.php';
$smarty = new Smarty;

$list = array('Books', 'Are', 'Fun');
$smarty->assign("list", $list);

$smarty->display('index.tpl');
?>
```

The output of this script will be:

```
<b>Books</b><br>
<b>Are</b><br>
<b>Fun</b><br>
```

The same thing is valid for associative arrays, but somewhat different:

```
<?php
require 'Smarty.class.php';
$smarty = new Smarty;

$list = array('name' => 'Tim',
        'last_name' => "O'Reilly"
);
$smarty->assign("list", $list);

$smarty->display('index.tpl');
?>
```

The template source will be somewhat different for that associative array:

```
{if $list != ""}
  <b>Name: {$list.name}; Last Name: {$list.last_name}</b><br>
{else}
  <b>Array $list has no entries</b>
{/if}
```

Some Real-World Examples

One of my favorite ways to use Smarty is to develop my applications with `PEAR::DB` and have templates take care of the layout formatting. Most of my applications nowadays are full of instances of the following:

```
<?php
include_once('DB.php'); // include PEAR::DB class
$db = DB::connect();

$stmt = "SELECT name, last_name FROM users";

// returns a nested associative array
$users = $db->getAll($stmt, DB_FETCHMODE_ASSOC);

include_once('Smarty.class.php');
$tpl = new Smarty;

$tpl->assign('users', $users);
$tpl->display('index.tpl');
?>
```

And the appropriate `index.tpl` would be:

```
{section name="i" loop=$list}
  <b>Name: {$list[i].name}; Last Name: {$list[i].last_name}</b><br>
{sectionelse}
  <b><br>Array $list has no entries</b>
{/section}
```

Smarty is so convenient and easy to use that it almost makes me sad thinking about all the time I used to spend doing `set_var()`, `parse()`, and `etc` when I used to use PHPLIB-based templates.

Another good example of how Smarty makes your life easier is the collection of built-in custom functions that come with it.

A simple yet very convenient set of functions is the `{html_options}`, `{html_select_date}`, and `{html_select_time}` tags. With these functions you don't need to worry about constructing HTML

select boxes in your PHP code. Rather, you can leave that boring task to Smarty, and it will format the output the way you wanted:

```
<select name="player_id">
        {html_options options=$players selected=$current.player_id}
</select>
```

With the template above all you need to do in your PHP code is the following:

```
<?php
include_once('Smarty.class.php');
$tpl = new Smarty;

$players = array(
    "23" => "Michael Jordan",
    "32" => "Michael Johnson"
);
$tpl->assign('players', $players);

$current_player = array(
    "player_id" => "23",
    "name" => "Michael Jordan"
);
$tpl->assign('current', $current_player);

$tpl->display('index.tpl');
?>
```

In my next article, I'll get to more in-depth coverage of the available features on Smarty as well as more examples of how to make the most out of it. See you then.

Joao Prado Maia is a web developer living in Houston with more than four years of experience developing web-based applications and loves learning new technologies and programming languages.

Return to PHP DevCenter.