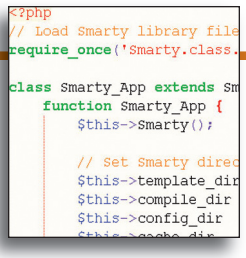
Getting Started with PHP and Smarty



by Ryan Nauman

Introduction

The language PHP (PHP: Hypertext Preprocessor) [1] has been around since 1995. Then, in early 2001, Smarty [3] was created. You may be asking yourself, "What is Smarty, and why should I use it?" Smarty is a template engine commonly used for Web applications written in PHP. Smarty's purpose is to separate a site's logic from its look and feel. It is an extremely valuable time-saving tool when used properly. If you use PHP, I strongly recommend using Smarty. Read on to find out why and if it is right for you.

Smarty requires PHP (4.0.6 or higher) and a Web server. This guide assumes you have these installed already. It also assumes you possess a basic working knowledge of PHP and HTML.

The Problem

Modularity and layering are extremely important concepts in computing. The importance lies in the realization that things change. Computer technologies seldom lay stagnant, and this fact needs to be accounted for when designing applications that need to stand the test of time. When you design a Web site, or more importantly, a Web application, it would be nice to simply take a list of requests from a customer and fulfill them. However, this is rarely the end of it. Customers often come back with additional feature requests or a list of problems that need to be corrected. When this happens, code needs to be reformatted, and in some cases, completely restructured and rewritten.

The concept of modularity often is not extended to the realm of Web design. Web sites frequently are created without the intention of making future design changes. I often find myself being asked to redesign or modify the design of an existing Web page that contains dynamically generated content. This can be quite a headache, even for small Web sites, when I find that HTML has been stuffed inside of PHP code. See Example 1 below.

Example 1: Sample bad PHP code.

```
<?php
$html = '<table>';
$html .= '<th>ID</th><th>Name</th><th>Address</th>
   <th>Code</th>';
foreach $users as $element {
   $html .= '<td>' . $element['id'] . '</td>';
   $html .= '<td>' . $element['name'] . '</td>';
   $html .= '<td>' . $element['address'] . '</td>';
   $html .= '<td>' . $element['code'] . '</td>';
}
$html .= '</table>';
?>
```

Many Web applications today lose their appeal over time if the design never changes. Redesigning a medium-to-large site can be a difficult process that leaves the designer stressed out and overwhelmed, ultimately leading to the redesign being abandoned. It starts to become especially difficult when PHP code is present and the original author has embedded HTML inside their PHP code. This is where Smarty comes in handy. Let's change this! See Example 2 and Example 3 below.

Example 2: Sample PHP code with Smarty.

```
<?php
$smarty->assign('user_info', $users);
$smarty->display('users.tpl');
?>
```

Example 3: Sample Smarty template code.

```
{html_table loop=$user_info cols="ID,Name,Address,Code"}
```

The Solution

Separate the Web application's logic code (PHP) from its design code (HTML)! Using Smarty, you can powerfully and easily take dynamic content and put it inside of your HTML without worrying about harming the data or the way the application works.

Smarty, like other template engines, relieves the pressure from the designer of harming application code. The designer can instead focus on the design of the site while still accessing dynamic content. Smarty is different from other template engines in that it is capable of much more than replacing tags. It provides support for features including but not limited to: debugging, compiling, extendability, security, and caching [4].

How Does Smarty Work?

Smarty pages are template files consisting of HTML and Smarty syntax. It looks very similar to a typical HTML page with the exception of accessing variables and functions using Smarty.

The process of displaying a Smarty page is as follows:

1. Check for caching.
2. Check for forced compilation.
3. Check for compiled template.
4. Compile template.
5. Generate output.
6. Send results to client.

Generally, the more features you find, the more overhead that is required. Therefore, it is natural if you find yourself concerned with the speed at which Smarty runs. However, there is no need to worry about the speed, because Smarty loads plug-ins as needed and compiles your template files only once (or when changed) [5].

Setting Up Smarty

Regular Smarty Installation

If you are running your Web server locally or if you have the ability to modify your php.ini file, then follow the steps below accordingly. If not, you can still use Smarty, but you will have to skip to the limited privileges installation steps in the next section.

1. Download Smarty from its official Web site [3].
2. Extract the libs folder from your archive to your Web server lib directory. If possible, Smarty should be placed outside the site's root for potential security reasons.
   a. Linux/Unix—/usr/local/lib/Smarty-v.e.r./
   b. Windows—c:\webroot\libs\Smarty-v.e.r.\
3. Change your php.ini file and add Smarty to your include_path.

Example 4: Sample php.ini *file.*

```
;;;;;;;;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;;;;;;;

; UNIX: "/path1:/path2"
;include_path = ".:/usr/share/php:/usr/local/lib/
;Smarty-v.e.r/libs/"
;
; Windows: "\path1;\path2"
include_path = ".;c:\php\includes;c:\webroot\libs\
   Smarty-v.e.r\libs\"
```

Now you are ready to create a Smarty object:

Example 5: Test file test.php.

```
<?php
require_once('Smarty.class.php');
$smarty = new Smarty();
?>
```

Run this script. If you have no errors, congratulations! If you run this script and encounter an error, double check your include_path using phpinfo();. If this is correct, then you may need to follow the Limited Privileges Smarty Installation setup.

Limited Privileges Smarty Installation

This section is intended for users that have limited privileges to their Web server. Skip these steps if you were able to follow the regular Smarty installation successfully.

1. Download Smarty from its official Web site [3].
2. Extract the libs folder from your archive to your Web server lib directory. If possible, Smarty should be placed outside the site's root for potential security reasons.
   a. Linux/Unix—/usr/local/lib/Smarty-v.e.r./
   b. Windows—c:\webroot\libs\Smarty-v.e.r.\

Now you are ready to create a Smarty object:

Example 6: Test file test.php.

```
<?php
// NOTE: Choose the appropriate style based on your
// webserver machine's Operating System

// unix or linux style (note capital 'S')
// define('SMARTY_DIR',
// '/usr/local/lib/Smarty-v.e.r/libs/');

// windows style
define('SMARTY_DIR', 'c:\webroot\libs\
   Smarty-v.e.r\libs\');

require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();
?>
```

Open this script in a browser. Congratulations, you have created your first Smarty object! There are a few more steps we need to complete before Smarty can begin to unleash its full potential.

Directories and Permissions

Now that we have the Smarty library files extracted and functioning, we need to set up a few folders on the site's root directory (or the directory from which you are using Smarty). There are four directories that are required. They are as follows:

1. cache—If enabled, Smarty stores cached versions of the pages here.
2. configs—Stores the configuration files used by the templates.
3. templates—This is where the templates are placed.
4. templates_c—This is where Smarty compiles template pages.

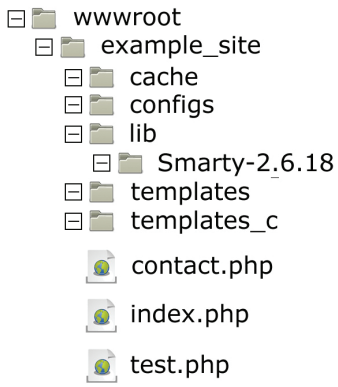


Figure 1: A sample directory structure.

These directory names can be changed if necessary, but those are the defaults. See Figure 1 for an example.

Smarty needs to be able to write to the cache and templates_c directories. The Web server account is what will actually be writing files to these directories. Windows users usually are not affected by this. However, in some cases, the directories need to be set up to have write access to the 'anyone' group. If you are on a Unix or Mac machine you will need to use chmod (inside a terminal window). Note that option 770 provides tight security, but 775 can be used instead for convenience, allowing the developer to view these files freely. See Example 7 below.

Example 7: Example chmod *commands.*

```
// The webserver usually uses the nobody user

// This command has to be done by the superuser.
// sudo might be necessary
chown nobody:nobody /www/example_site/cache/
chmod 770 /www/example_site/cache/

chown nobody:nobody /www/example_site/templates_c/
chmod 770 /www/example_site/templates_c/
```

Your Smarty setup is complete! That wasn't so bad now, was it? It may seem like a lot at first, but once you have done it a few times, it's a breeze. Now, let's move on to our first Smarty template file.

Creating a Template File

There are two basic components to any Smarty page: the template file and a PHP file. Let us first create our template file. It needs to be created in our templates folder. See Example 8 below.

Example 8: An example templates/index.tpl *file.*

```
// Test a template comment
{* Smarty *}

Hello world!  {$name} is now successfully
   running Smarty!
```

Take a moment to note the syntax used by Smarty. Smarty code is simply entered between curly bracket characters ({}). Later on, you will see that template files are nothing more than an HTML page with Smarty syntax inside it.

Now, let's move on to our PHP file that will use this template created in the previous step. Create this PHP file in your Web site's root. See Example 9 below.

You are now officially a Smarty user! You may have begun to see some of the possibilities of Smarty as we moved through these initial steps. Rest assured that this is only the beginning of what Smarty can do. Read on to learn some of the more advanced and useful techniques.

Advanced Smarty Setup

The way you were shown to setup Smarty as mentioned earlier is fine, and it works. However, do you really want to have to instantiate a new Smarty object every time you create a new template file? No way! Let's go over the advanced setup procedure.

Example 9: An example index.php file.

```
<?php

require_once('Smarty.class.php');

// Limited installation users ONLY:
// Uncomment the line below and remove the line above
// require_once(SMARTY_DIR . 'Smarty.class.php');

// Create our Smarty object
$smarty = new Smarty();

// Set the Smarty directories
$smarty->template_dir = '/wwwroot/example_site/
   templates/';
$smarty->compile_dir = '/wwwroot/example_site/
   templates_c/';
$smarty->config_dir   =
   '/wwwroot/example_site/configs/';
$smarty->cache_dir    = '/wwwroot/example_site/cache/';

// Assign a value to a smarty variable
$smarty->assign('name', 'Bernard');

// Display our template
// We do not need to specify that it is in 'templates/'
// because we have assigned the Smarty directories
$smarty->display('index.tpl');

?>
```

I like to create an init.php file in a folder called lib under my Web site's root containing all of my site's initializations. This usually includes things like the site's name, database information, and a Smarty object instantiation. See Example 10 below.

That may have been a lot to swallow at once. If you are unfamiliar with the PHP portions of this file, you should reference the official PHP online documentation [2]. Let us review the Smarty-related code.

Smarty is nice in that it allows us to extend its class. This makes declaring a Smarty object easier to understand and more pleasing to read. We then create a function with the same name as the class (a constructor). This special function will be called every time an object of the Smarty_DefaultSite class is created. Next, we have some simple directory declarations. Here we can also assign a global site or application name to be easily included in the title of every page. Next, we enable the debugging. This is an especially important feature used in development. Smarty comes packaged with a debugging console that will let you know every piece of data (from our PHP pages that call the template) that has been assigned to Smarty variables. To use it, put a parameter in the URL called SMARTY_DEBUG. Example: http://www.defaultsite.com/index.php?SMARTY_DEBUG or http://www.defaultsite.com/show.php?id=25&SMARTY_DEBUG.

Example 10: An example lib/init.php *file.*

```
<?php

// Include some commonly used custom functions
require_once('functions.php');

// Database information
$db = array (
    'host' => 'localhost',
    'user' => 'user',
    'pass' => 'pass',
    'name' => 'db_name'
);

// Error reporting, site root
error_reporting(E_ALL);
$SITE_ROOT = '/wwwroot/example_site/';

// Grab the Smarty files
require_once('Smarty.class.php');

// Extend the Smarty object
class Smarty_DefaultSite extends Smarty {

    function Smarty_DefaultSite()
    {
        // Class Constructor.
        $this->Smarty();

        // Set Smarty directories
        $this->template_dir = $SITE_ROOT . 'templates/';
        $this->compile_dir  = $SITE_ROOT . 'templates_c/';
        $this->config_dir   = $SITE_ROOT . 'configs/';
        $this->cache_dir    = $SITE_ROOT . 'cache/';

        // Assign our application's name
        $this->assign('app_name', 'Default Site');

        // Enable debugging via URL
        // This is recommended for development
        // phases only
        $smarty->debugging_ctrl = 'URL';

    }

}

// I tend to use Smarty on every page
// if I am going to use it at all
// So let's write less code and instantiate
// our Smarty object now
$smarty = new Smarty_DefaultSite();

?>
```

After we have created this file, we can simply use require_once ('lib/init.php') on every PHP page that is created. We are now free to assign variables and display Smarty templates at will. Read on to find out the incredible Smarty functions we can use inside our template files.

Key Smarty Functions

This is the part of the guide where key functions essential to Smarty's power are finally revealed. Let's learn how to migrate away from embedding HTML inside our PHP files as so many of us have been guilty of doing in the past.

Note that these examples assume you are comfortable with PHP. Covering PHP programming is beyond the scope of this tutorial, and should you need help, there is plenty of documentation available at php.net [2].

Function: {include}

If you are familiar with PHP, then you are aware of the include function. Well, Smarty has one too that is equally essential. You use the include function in Smarty for including other template files, such as a navigation menu. Let's take a look at some example template code. See Example 11 and Example 12 below.

Example 11: The navigation.tpl *file.*

```
<div id="nav">
  <a href="members.php" title="Members">Members</a>
  <a href="articles.php" title="Articles">Articles</a>
  <a href="news.php" title="News">News</a>
  <a href="index.php" title="Home">Home</a>
</div>
```

Example 12: The index.tpl *file.*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
   xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta http-equiv="Content-Type"
      content="text/html;charset=utf-8" />
   <title>Home</title>
</head>
<body>

{include file='navigation.tpl'}

<div>
   <p>Welcome!</p>
</div>

</body>
</html>
```

Function: {if}, {elseif}, {else}

Suppose that on our site's main page we have a welcome portion. Maybe we want to prompt the user to log in if they have not already. And if the user is logged in, let's greet them personally with their

name. We'll first take a look at what our PHP file might look like. See Example 13 below.

Example 13: An example index.php *file.*

```
<?php

// Grab our initialization file with our Smarty
// object defined
require_once('lib/init.php');

// Assuming we have code inside our functions.php file,
// let's see if the user is logged in
$is_logged_in = ($user->isLoggedIn()) ? true : false;

// Now if they are logged in let's grab their name
// and assign it
if ($is_logged_in) {
    $smarty->assign('name', $user->getUserName());
}

// Display our template
$smarty->display('index.tpl');

?>
```

Now let's create our index.tpl file. See Example 14 below.

Example 14: An example index.tpl *file.*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
    Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
    xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <title>{$app_name} - Home</title>
</head>
<body>

{if ($is_logged_in)}
    <h1>Welcome {$name}!</h1>
{else}
    <h1>Welcome! Please <a href="login.php">login
        </a></h1>
{/if}

</body>
</html>
```

You can see that our template file is identical to HTML except for the Smarty variables enclosed in curly bracket ({}) characters.

Smarty allows you to use any PHP function inside the condition part of the if statement. For example: {if is_array($scalar)}.

Let's revisit examples 13 and 14 using a more object oriented (OO) approach. See Example 15 below.

Note: From this point on, all obvious HTML code will be removed (e.g., DOCTYPE, head, body, etc.). The focus will be solely on the Smarty code.

Example 15: An OO example of the index.php *file.*

```
<?php

// Grab our initialization file with our Smarty
// object defined
require_once('lib/init.php');

// Let's hand the user object over to Smarty
// and let it handle the logic
$smarty->assign('user', $user);

// Display our template
$smarty->display('index.tpl');

?>
```

Now let's see how Smarty handles objects. Feel free to make use of the URL debugging option we enabled earlier, in order to visualize what is happening. See Example 16 below.

Example 16. An OO example of the index.tpl *file.*

```
{if ($user->isLoggedIn())}
    <h1>Welcome {$user->getUserName()}!</h1>
{else}
    <h1>Welcome! Please <a href="login.php">
        login</a></h1>
{/if}
```

Smarty handles objects with the same syntax that PHP does. Let's see what Smarty can do with something a little more complex, such as an array of objects.

Function: {foreach}, {foreachelse}

Let's say that we have an array of objects. A sample scenario would be a list of users and their information. Maybe we wish to iterate over the array of user objects and display a nice list with some of their information, such as their name and ID.

As usual, we will start with the PHP code first and then move on to the Smarty template code. Go ahead and set things up by creating a PHP file on your site named 'users.php'. See Example 17 below for the code.

Now we have an array of objects (*hopefully!*), and we are going to build our template file to work with it. Create a file named 'users.tpl' inside your templates folder. Example 18 below shows the code.

Let us break down what is happening in our template file, users.tpl, shown below:

- The from attribute tells the loop which Smarty variable to loop over. Note that we must use the dollar sign ($) here to indicate this is a smarty variable.

Example 17: An OO example of the **users.php** *file.*

```
<?php
// Grab our initialization file
require_once('lib/init.php');

// Let's assume we have a function that
// connects to our db
$link = getConnection( );
$query = "SELECT f_name, l_name, user_id
    FROM users ORDER BY l_name ASC";

$result = mysql_query($query, $link);

$counter = 0;
while ($row = mysql_fetch_assoc($result) {
   // Assume our User class has a constructor
   // that accepts
   // these parameters and assigns the object's properties
   $users[$counter] = new User($row);
   $counter++;
}

// Assign the array of objects to a smarty var
$smarty->assign('users', $users);

?>
```

Example 18: An OO example of the **users.tpl** *file.*

```
<h1>List of Users</h1>

{foreach from=$users item=user name=user_loop}
    {if $smarty.foreach.user_loop.first}<ul>{/if}
    <li>{$user->getLastName( )}, {$user->getFirstName()} –
        <strong>{$user->getUserId( )}</strong></li>
    {if $smarty.foreach.user_loop.last}</ul>{/if}
{foreachelse}
    <p>I'm sorry, no users were found.</p>
{/foreach}
```

- The **item** attribute lets the loop know what we're going to refer to each array element as inside the loop. I like to always pluralize my array name and refer to each item as the singular version inside my loops.
- The **name** attribute is what lets us access the powerful foreach properties, such as first and last, as seen above.
  - **first** will return true when the foreach loop is in its first iteration.
  - **last** will return true when the foreach loop is in its last iteration.
- **foreachelse** is handy in case the array of objects we thought we assigned in our PHP code turns out to be empty.

Some things to note about **foreach** loops:

- **foreach** loops can be nested.
- **foreach** loops can cycle through associative arrays or numerically indexed arrays.
- **foreach** loops can also handle key-value arrays (hashes).

See Example 19 and Example 20 below for an example of the foreach and a key based array.

Example 19: Example of a key-value array.

```
<?php

$my_array = array( 0 => 'Tom',
                   1 => 'Ralph',
                   2 => 'Amy',
                   3 => 'Herman'
                 );

$smarty->assign('my_array', $my_array);

?>
```

Now take a look at the template code below.

Example 20: An example of a foreach loop with a key-value array.

```
<h1>List of Users</h1>

{foreach from=$my_array key=my_key value=
    my_value name=user_loop}
    {if $smarty.foreach.user_loop.first}<ul>{/if}
    <li>$my_key: $my_value</li>
    {if $smarty.foreach.user_loop.last}</ul>{/if}
{foreachelse}
    <p>I'm sorry, no users were found.</p>
{/foreach}
```

Conclusion

If you have made it this far, congratulations! You should be able to download and place required Smarty library files, create Smarty objects inside PHP code, and write your own Smarty template files. Now get out there and start incorporating Smarty into any PHP applications you develop from now on!

Further Reading

If you have mastered the above techniques and are still craving more, I strongly encourage you to read over the documentation listed on Smarty's official Web site [3]. Their guides are not as friendly or geared towards beginners as this one is, but if you've made it this far, I am sure you can handle it!

Some other functions I omitted from this tutorial that I strongly suggest going over are: **{section}**, **{eval}**, **{capture}**, **{html_*}**, and the **date** functions. You can even learn to create your own functions should you find the need.

Some of these examples as well as real-world and more complicated examples are posted on the Web at ryannauman.com under the

Smarty section. You can also view live demos and view the source code powering them.

References

1. PHP: Hypertext Preprocessor. 2007. http://www.php.net/ (accessed October 2, 2007).
2. PHP: PHP Manual—Manual. 2007. http://www.php.net/manual/en/ (accessed October 2, 2007).
3 Smarty: Template Engine. 2007. http://smarty.php.net/ (accessed October 2, 2007).
4. Smarty. 2007. http://smarty.php.net/manual/en/what.is.smarty.php (accessed October 2, 2007).
5. Smarty. 2007. http://smarty.php.net/manual/en/variable.compile.check.php (accessed October 2, 2007).

Biography

Ryan Nauman (nau8019@cup.edu) graduated from California University of Pennsylvania in December 2007 with a degree in Information Technology.