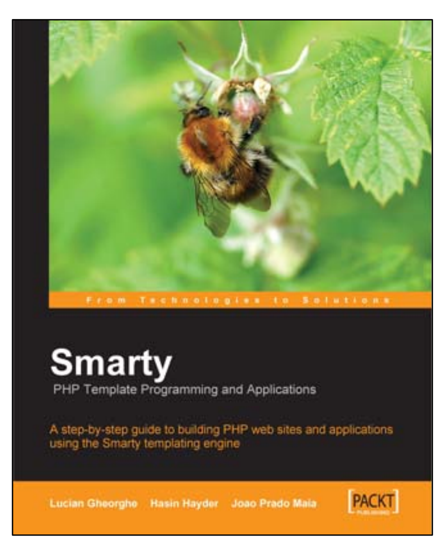
Smarty:

PHP Template Programming and Applications

João Prado Maia
Hasin Hayder
Lucian Gheorghe



Chapter 6
"Smarty Functions"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter 6 "Smarty Functions"

A synopsis of the book's content

Information on where to buy this book

About the Authors

João Prado Maia is Lead Software Developer with Alert Logic, Inc. and was previously with MySQL AB as the lead developer behind Eventum, an issue tracking system, and MySQL Network, a subscription product for everything related to MySQL services. He has been working with PHP, Smarty, and PEAR for several years, and maintains phpbrasil.com, one of the most popular PHP-related community sites in Brazil. He is also interested in fostering a community of PHP developers in Houston by organizing the Houston PHP Users Group at `http://houstonphp.org`.

Hasin Hayder graduated in Civil Engineering from Rajshahi University of Engineering and Technology (RUET). He is an open-source enthusiast who has been programming since early 2001. He maintains phpXperts, the largest PHP user group in Bangladesh, and Zephyr, an open-source AJAX-based MVC framework for PHP5 developers. He is currently working as a web application developer in a Norwegian software development company, "Somewhere In...". You can reach him at `hasin@somewherein.net`.

Lucian Gheorghe is currently working as a senior network engineer for Globtel Internet, a significant Internet and Telephony Services Provider to the Romanian market. Even if it's not his main activity, He has been programming in PHP for over 5 years building billing interfaces, industrial software interfaces, e-commerce sites, and so on. He had a lot of help from a friend called Smarty in his programming experiences.

Lucian got his first taste of writing when he contributed a few chapters to the book *Beginning PHP 5 and MySQL E-Commerce* by Cristian Darie and Mihai Bucica, Apress, 2004, with his appendix for Project Management added to the book *Beginning ASP.NET 2.0 E-Commerce in C# 2005* by Cristian Darie and Karli Watson, Apress, 2005.

6

Smarty Functions

Smarty provides a wide variety of functions available to you as a template designer to use as part of your templates. While you may never use some of these functions, it is important to get to know their functionality, as they were created with the ultimate goal of solving everyday problems of web developers.

As a rule of thumb, if you need some functionality in your templates and you don't know for sure if Smarty already provides it, check if there is already a function for it. If you require some custom functionality, consider writing a Smarty plug-in.

In this chapter you will learn about the different types of functions available in Smarty, a list of those functions and examples of their use, and how to extend Smarty and create your own functions.

Types of Smarty Functions

Smarty as a software project tries to be simple and this is seen in the built-in functionality that is shipped by the template compiler and parser. The template language is itself built on functions that enable the basis of the functionality of a template, and you can extend the standard functionality of Smarty with plug-ins. However, the built-in functions cannot be modified and you cannot create custom functions with the same names.

Since the development team knew that they couldn't predict all uses of Smarty, or all needed functions, they decided to create a plug-in architecture and let the end users create, modify, and submit new plug-ins. Most of the Smarty functions are themselves implemented using this very plug-in architecture.

Functions in Action

We will build a relatively complex site in this chapter, while explaining step by step the strengths and weaknesses of particular functions. At the end of this chapter, you should be able to choose the appropriate functions when you need some help from Smarty.

Let's start this process with a simple web page, a front page for a company website, along with its template.

For More Information: www.packtpub.com/smarty/book

index.tpl

```
<html>
<head>
<title>Smarty LLC</title>
</head>
<body>
<h3>Smarty LLC</h3>
<p>We provide consulting services to the healthcare industry.</p>
<p>For more details, please contact us by email on contact@smartyllc.com</p>
</body>
</html>
```

index.php

```
<?php
include_once('libs/Smarty.class.php');
$smarty = new Smarty;
$smarty->display('index.tpl');
?>
```

This is a simple website, with just one page. Let's expand this by creating a couple of different pages, one listing the management team of our company, and another displaying a separate screen with a contact form instead of telling potential customers to send an email to a particular address.

Here's the page listing the management team:

about.tpl

```
<html>
<head>
 <title>Smarty LLC</title>
</head>
<body>

<h3>Smarty LLC: Management Team</h3>

<p><b>Julian Fox, CEO</b> - Julian handles all management issues related to
the company, such as dealing with potential customers and providing a clear
strategy for services.</p>

<p><b>Ryan Foster, CTO</b> - Ryan handles all technical aspects of consulting
services available through Smarty LLC.</p>

</body>
</html>
```

And the following will be the new contact page. We will not provide the PHP code to actually send out emails from the contact form, but that's simple and can be looked up on the Internet.

contact.tpl

```
<html>
<head>
 <title>Smarty LLC</title>
</head>
<body>

<h3>Smarty LLC: Contact Us</h3>
```


For More Information: www.packtpub.com/smarty/book

```
<form method="post" action="contact_us.php">
<p>
 <b>Your Name:</b><br />
 <input type="text" name="name" size="40">
</p>

<p>
 <b>Your Email:</b><br />
 <input type="text" name="email" size="40">
</p>

<p>
 <b>Your Message:</b><br />
 <textarea name="message" cols="40" rows="10"></textarea>
</p>

<input type="submit" value="Send Message">
</form>

</body>
</html>
```

So now we have a slightly more complicated website, and some challenges are starting to appear. Even though our collection of pages is still small, every change in a section of the site that is used by all of these pages will require three different changes. Let's use some of Smarty features to help us with this maintenance headache.

Action: Re-using Page Elements with the include Function

1. Create the following file and save it as `header.tpl` in your template directory:

   ```
   <html>
   <head>
    <title>Smarty LLC</title>
   </head>
   <body>
   ```

2. Create another file called `footer.tpl` in the same location, with the following content:

   ```
   </body>
   </html>
   ```

3. Now make the following changes to your existing web pages:

   - Replace the following HTML snippets:

     ```
     <html>
     <head>
     <title>Smarty LLC</title>
     </head>
     <body>
     ```

     With:

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

   - Substitute the following:

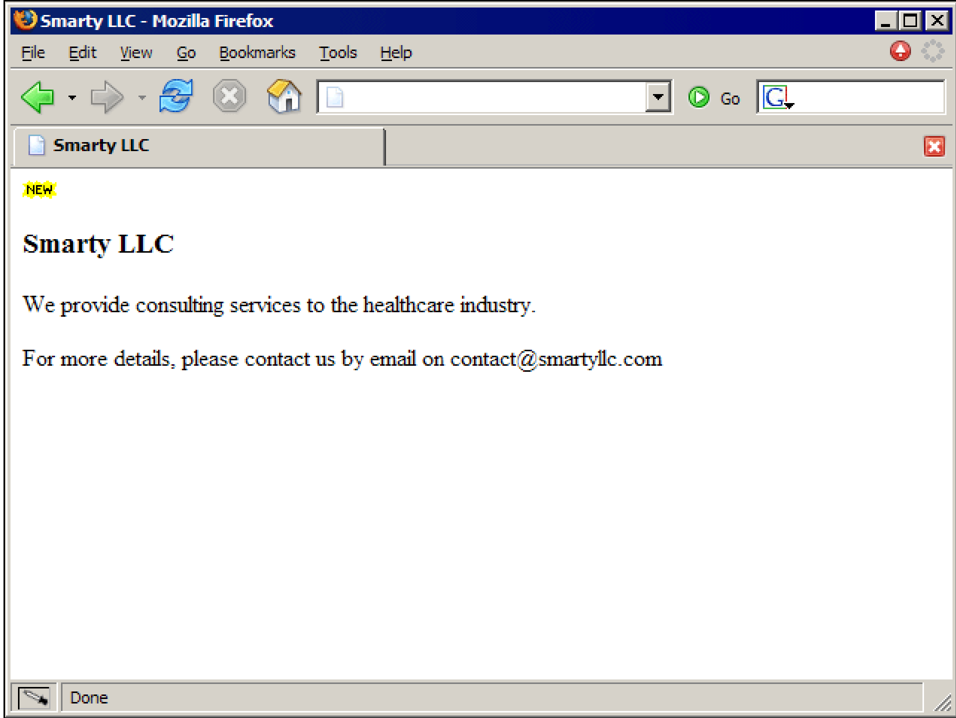
     ```
     </body>
     </html>
     ```

     With:

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


For More Information: www.packtpub.com/smarty/book

4. Open your web browser and take a look at your website again. You should see exactly what you were seeing before. Try adding an image tag to add the image `new.jpg` to your `header.tpl` template file, and reloading the website in your browser. You should see the new image on all the web pages, as shown:



Explanation

The interesting thing about the `include()` function is that it is a pretty powerful way of modularizing your templates, and easing the maintenance work that you must do in the future. Instead of changing all three pages when you want something changed in the header portion of your website, all you need to do is update `header.tpl`.

Inserting Dynamic Content

Let's add some more details to our website by displaying the current date and time at the headquarters of Smarty LLC. While we could use the same implementation we used above with the `include()` function, we will instead use the `insert()` function.

This function works in almost the same way as `include`, with a few important differences:

- The results are not going to be cached by Smarty. This is an important detail in some circumstances, such as when you need to be absolutely sure that some portions of your templates are always dynamically generated.

For More Information: www.packtpub.com/smarty/book

- You are not actually including a file, but running a PHP function that you will need to provide. The `name` attribute is what tells Smarty what function it should call. For example, if you pass `getCurrentTime` to it, Smarty will call a function called `smarty_insert_getCurrentTime`.

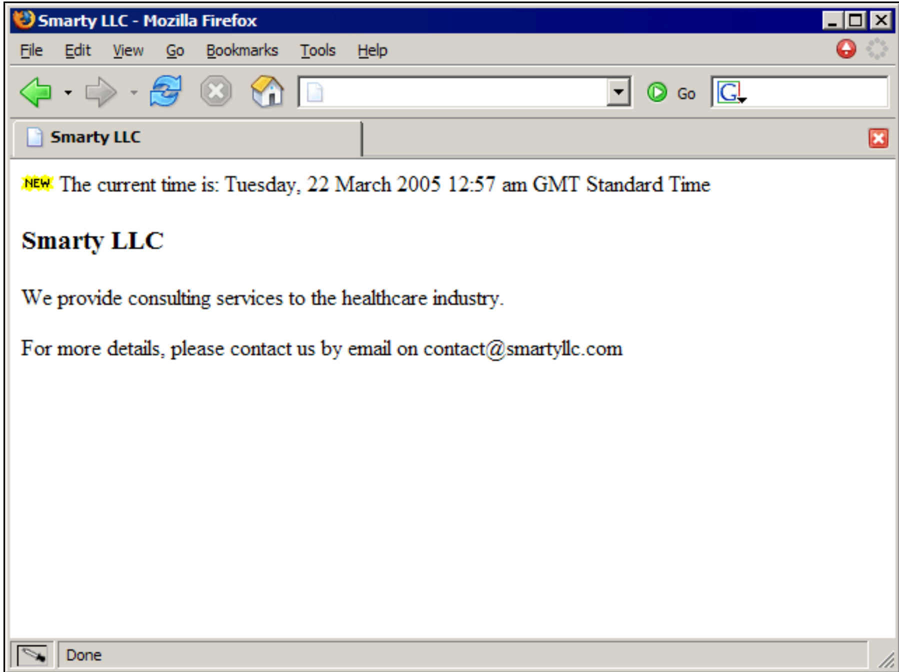
You may also assign the output of this function to a template variable, and even call a special PHP script prior to executing it. Here's what our `header.tpl` file looks like now with the changes:

```
<html>
<head>
 <title>Smarty LLC</title>
</head>
<body>

<img src="images/new.gif">

{insert name="getCurrentTime" assign="current_time"
script="time_functions.inc.php"}

The current time is: {$current_time}
```

And the code for `time_functions.inc.php` is:

```
<?php
function smarty_insert_getCurrentTime()
{
    return gmdate('l, j F Y g:i a T');
}
?>
```

Here's what the front page of our website looks like now:



For More Information: www.packtpub.com/smarty/book

Passing Variables to Included Templates

Another very useful feature of the `include()` function is the ability to pass any number of arguments to the included templates. That allows you to include a template and override some default behavior when passing a flag to the inner template. So instead of having to display the current page title on each of our templates, let's create a new template file called `navigation.tpl` and use this special feature. Here are the contents of the new `navigation.tpl` file:

```
<h3>Smarty LLC{if $page_title != ''}: {$page_title}{/if}</h3>
```

Instead of simply using the include function to add the contents of this file to your other templates, we will also pass a variable to it. Here are the new contents of `index.tpl`:

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

<p>We provide consulting services to the healthcare industry.</p>

<p>For more details, please contact us by email on contact@smartyllc.com</p>

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

Change the `about.tpl` template to the following:

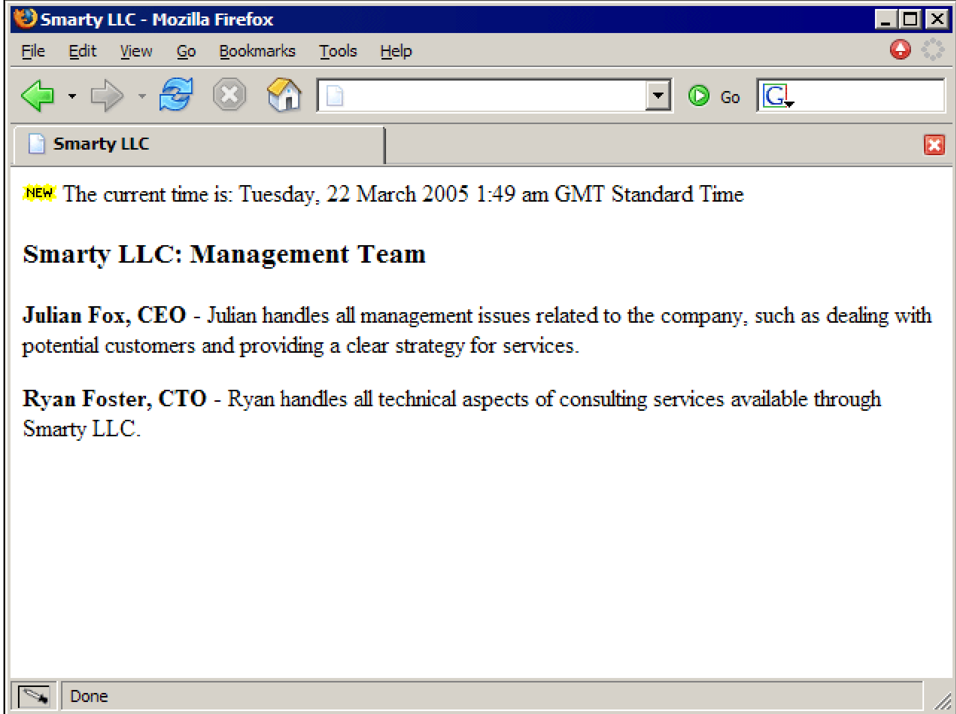
```
{include file="header.tpl"}
{include file="navigation.tpl" page_title="Management Team"}

<p><b>Julian Fox, CEO</b> - Julian handles all management issues related to
the company, such as dealing with potential customers and providing a clear
strategy for services.</p>

<p><b>Ryan Foster, CTO</b> - Ryan handles all technical aspects of consulting
services available through Smarty LLC.</p>

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

And change `contact.tpl` to have the following content:

```
{include file="header.tpl"}
{include file="navigation.tpl" page_title="Contact Us"}

<form method="post" action="contact_us.php">
<p>
 <b>Your Name:</b><br />
 <input type="text" name="name" size="40">
</p>

<p>
 <b>Your Email:</b><br />
 <input type="text" name="email" size="40">
</p>

<p>
 <b>Your Message:</b><br />
 <textarea name="message" cols="40" rows="10"></textarea>
</p>

<input type="submit" value="Send Message">
</form>

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


For More Information: www.packtpub.com/smarty/book

Perform the same change across all other template files, and open the 'about' page in your browser. You should see something like the following:



This looks exactly like what you had before, except that we are now modularizing our templates even more. The good thing now is that we can expand our new template with bread-crumb style navigation (that is, Home | Section | Sub-Section) in an easy way, by changing only one file.

Saving Variables in Configuration Files

It is extremely easy to pass variables to templates by using the normal PHP-based `assign()` function, like the following:

```
<?php
include_once('libs/Smarty.class.php');
$smarty = new Smarty;

$smarty->assign('company_name', 'Smarty LLC');

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

That would allow you to use the variable `$company_name` in your templates. While this is very handy, it doesn't make much sense to have that same line in each of your PHP scripts. Instead of duplicating PHP code, you can use Smarty configuration files to save variables and make them available to your templates.

For More Information: www.packtpub.com/smarty/book

A good example of the use of the `{config_load}` function is to translate your templates to several languages, but still keep one copy of your template structure for all of these separate sites.

Instead of having several copies of the same site structure, each having its own templates, but with hard-coded strings in each language, you can have just one site structure, and several configuration files, one for each language.

Create a file named `english.conf` and save the following content in it:

```
# global variables
company_name = "Smarty LLC"
```

Now change your existing `navigation.tpl` template file to this:

```
{config_load file="english.conf"}

<h3>{#company_name#}{if $page_title != ''}: {$page_title}{/if}</h3>
```

As you can see, you can use the variables saved in the configuration file directly in your templates.

Creating Configuration Sections for Each Page

While our existing `english.conf` file is pretty simple right now, it could grow quickly as we add more pages to our website, and as we translate each section of our site to separate languages. Instead of loading the entire configuration file as we did before, let's separate the variables of each page into sections, and load only the appropriate sections in our templates.

1. Modify the existing `english.conf` file to the following (the triple quotes are required for strings that span multiple lines):

   ```
   # global variables
   company_name = "Smarty LLC"

   [Index]
   intro_paragraph_1 = """We provide consulting services to the healthcare
   industry."""
   intro_paragraph_2 = """For more details, please contact us by email on
    contact@smartyllc.com"""
   ```

2. Change `navigation.tpl` back to the following:

   ```
   <h3>{#company_name#}{if $page_title != ''}: {$page_title}{/if}</h3>
   ```

3. Change `index.tpl` to the following:

   ```
   {config_load file="english.conf" section="Index"}
   {include file="header.tpl"}
   {include file="navigation.tpl"}

   <p>{#intro_paragraph_1#}</p>

   <p>{#intro_paragraph_2#}</p>

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


For More Information: www.packtpub.com/smarty/book

Firstly, notice how we pass the section name, in this case `Index` to the `{config_load}` function. This tells Smarty that we are only interested in the variables associated with that particular section. Also note that while we do specify the section, the global variable called `company_name` is still available in `navigation.tpl`. The reason is that this variable is global, and will always be available when that configuration file is loaded into the template.

The examples above set the configuration file manually to `english.conf`, but we could just as well expand on this and pass a variable to the template, and dynamically set the language of our website. It would be trivial to expand it even more and allow our visitors to choose the language of the site that they are interested in. Consider the following changes to `navigation.tpl`:

```
<p>
  Language:
  <a href="?language=en">English</a> |
  <a href="?language=it">Italian</a>
</p>

<h3>{#company_name#}{if $page_title != ''}: {$page_title}{/if}</h3>
```

We are showing two links to our visitors, and allowing them to reload the current page with a query string, `?language=en` or `?language=it`. With some tweaks to our existing PHP code, we will switch languages on the fly:

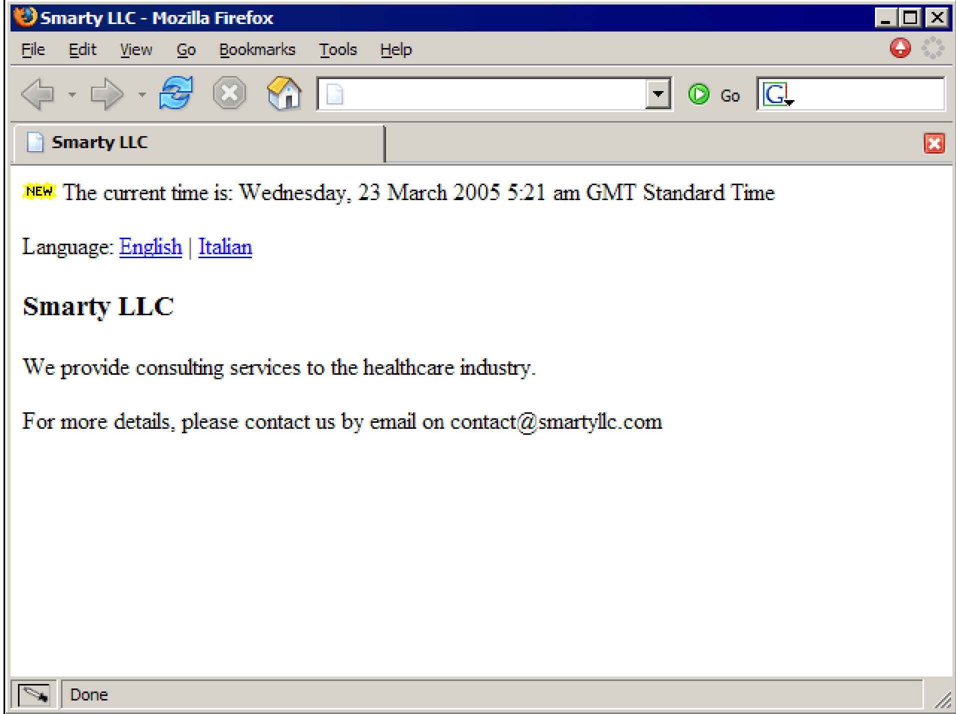
```
<?php
include_once('libs/Smarty.class.php');
$smarty = new Smarty;

if (empty($_GET['language'])) {
    $smarty->assign('language', 'english');
} else {
    if ($_GET['language'] == 'en') {
        $smarty->assign('language', 'english');
    } elseif ($_GET['language'] == 'it') {
        $smarty->assign('language', 'italian');
    }
}

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

Now change your existing `index.tpl` file to the following:

```
{config_load file="$language.conf" section="Index"}
{include file="header.tpl"}
{include file="navigation.tpl"}

<p>{#intro_paragraph_1#}</p>

<p>{#intro_paragraph_2#}</p>

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


For More Information: www.packtpub.com/smarty/book

Let's see how this looks:



We are now dynamically selecting the appropriate language for the configuration file. You just got yourself an easy-to-maintain international website with a few simple steps!

Handling Lists in Templates

While our templates now handle a lot of custom functionality, such as content translated into several languages, and dynamically displaying the page title based on what value we pass when including another file, the site is still pretty static. That is, there isn't a lot of content on the site that is not known at the time that each particular page is requested.

Instead of hard-coding the list of available languages in `navigation.tpl`, let's dynamically generate the list of languages from a PHP array passed to the template.

1. Change `index.php` to the following:

```
<?php
include_once('libs/Smarty.class.php');
$smarty = new Smarty;

if (empty($_GET['language'])) {
    $smarty->assign('language', 'english');
} else {
    if ($_GET['language'] == 'en') {
```


For More Information: www.packtpub.com/smarty/book

```
        $smarty->assign('language', 'english');
    } elseif ($_GET['language'] == 'it') {
        $smarty->assign('language', 'italian');
    }
}

// list of available languages
$languages = array(
    'en' => 'English',
    'it' => 'Italian',
    'de' => 'German',
    'pt' => 'Portuguese'
);
$smarty->assign('languages', $languages);

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

2. Now change `navigation.tpl` so that this array is processed and the HTML is dynamically generated according to the values we provide from the PHP script:

```
<p>
  Language:
{foreach from=$languages key="abbreviation" item="title"}
  <a href="?language={$abbreviation}">{$title}</a> |
{/foreach}
</p>

<h3>{#company_name#}{if $page_title != ''}: {$page_title}{/if}</h3>
```

3. Open up our website in your browser, and you should now see something similar to this:

For More Information: www.packtpub.com/smarty/book

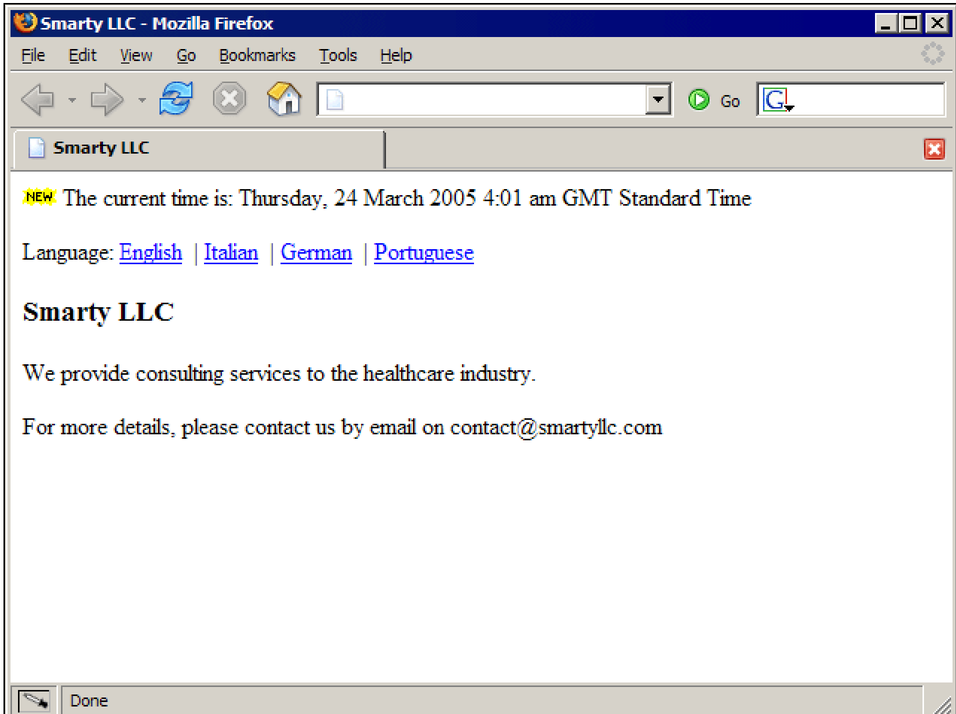
As you can see, the `{foreach}` function loops through the PHP array, and displays each of its options as a link on the page. The `key` attribute determines the name of the variable that will hold the key of the array element, and `item` will set the name of the variable that will hold the value.

There are several other attributes and features available through the `{foreach}` function. Notice how the list of languages on the previous screenshot ends with a space and the pipe character. Let's avoid adding those extra characters by using another feature of `{foreach}`.

1. Change `navigation.tpl` to include the following content:

```
<p>
  Language:
{foreach from=$languages key="abbreviation" item="title" name="lang"}
  <a href="?language={$abbreviation}">{$title}</a>
  {if not $smarty.foreach.lang.last}
   |
  {/if}
{/foreach}
</p>

<h3>{#company_name#}{if $page_title != ''}: {$page_title}{/if}</h3>
```

2. Open your browser again on the website, and notice that the layout has changed:



For More Information: www.packtpub.com/smarty/book

There you go! The last white space and pipe characters are not being displayed anymore. By setting the `name` attribute on the `{foreach}` call, we are able to reference back to this loop from within your templates. The special variable `$smarty.foreach.lang` is how you refer to this loop, and the `last` attribute will tell you whether you are at the last iteration of the loop or not.

The code basically checks whether Smarty is processing the last iteration of the loop, and will output the space and pipe characters if that is not the case.

However, if you pay enough attention to details, you must have noticed that now the navigation links have an extra white space before all the pipe characters. This is basically a side effect of how we generated the HTML for the navigation links, which should look similar to the following:

```
<p>
  Language:
  <a href="?language=en">English</a>
     |
    <a href="?language=it">Italian</a>
     |
    <a href="?language=de">German</a>
     |
    <a href="?language=pt">Portuguese</a>
</p>
```

You see, browsers will render a white space before the non breaking space. We can fix that with yet another built-in Smarty function.

Removing Extra White Space from Templates

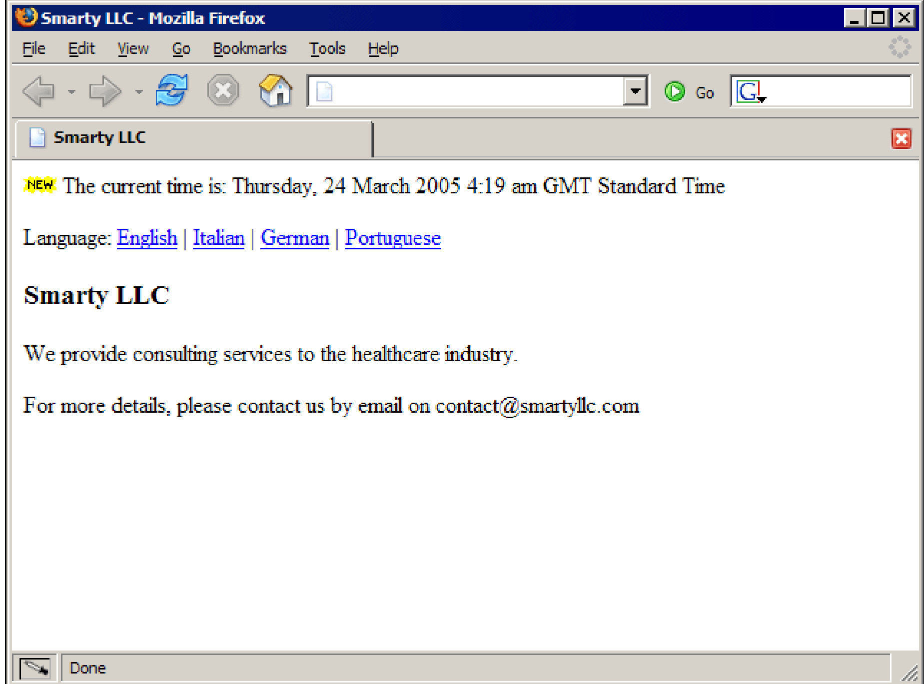
Let's change the `navigation.tpl` template again to remove the extra white space that ended up being displayed when we last modified our file.

1. Open the file `navigation.tpl` and change it to include the following content:

```
<p>
  Language:
{foreach from=$languages key="abbreviation" item="title" name="lang"}
  {strip}
  <a href="?language={$abbreviation}">{$title}</a>
  {if not $smarty.foreach.lang.last}
   |
  {/if}
  {/strip}
{/foreach}
</p>

<h3>{#company_name#}{if $page_title != ''}: {$page_title}{/if}</h3>
```

2. Reload the website again, and notice that the extra white spaces before the pipe characters are now gone:

For More Information: www.packtpub.com/smarty/book



3. View the source of this page, and the navigation-related section should look something like this:

```
<p>
  Language:
  <a href="?language=en">English</a>  |
<a href="?language=it">Italian</a>  |
<a href="?language=de">German</a>  |
<a href="?language=pt">Portuguese</a></p>
```

Notice how it's all in the same line now. The `{strip}` function is very useful for these types of cases in which white space matters in your pages, but it would be easier to maintain your website if you could still have it all on separate lines.

Handling JavaScript Code in Templates

It may seem like embedding JavaScript code in your templates might be straightforward, but a very important catch that you should be looking out for is that, since braces are used by Smarty as delimiters for function names and variables, something simple like creating a JavaScript function might result in a template parsing error. You can avoid this by following these steps.

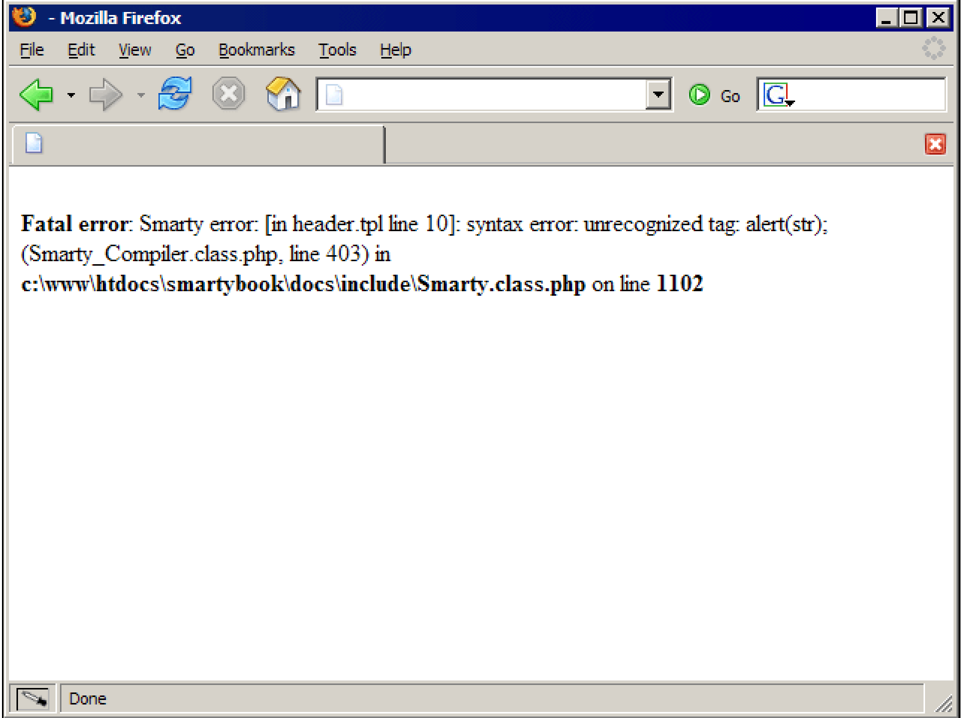
Change `header.tpl` to the following:

```
<html>
<head>
 <title>Smarty LLC</title>
</head>
```


For More Information: www.packtpub.com/smarty/book

```
<body>
<script language="JavaScript">
<!--
function printMessage(str)
{
    alert(str);
}
//-->
</script>
<img src="images/new.gif">
{insert name="getCurrentTime" assign="current_time"
script="time_functions.inc.php"}
The current time is: {$current_time}
```

Reload any page of our website (since all of our templates include `header.tpl` in them), and you should see an error message similar to the following:



As you can see from the screenshot above, Smarty has a problem understanding that the function `printMessage` is something related only to the JavaScript code, and should be ignored. In order to fix this, there are two solutions using Smarty functions available to you:

- The combination of the functions `{ldelim}` and `{rdelim}`. The first one outputs the left template delimiter, and the last one outputs the right delimiter, which happen to be left and right braces. This is used by modifying `header.tpl` to use the `{ldelim}` and `{rdelim}` functions, as follows:

For More Information: www.packtpub.com/smarty/book

```
<html>
<head>
  <title>Smarty LLC</title>
</head>
<body>
  <script language="JavaScript">
  <!--
  function printMessage(str)
  {ldelim}
  alert(str);
  {rdelim}
  //-->
</script>
<img src="images/new.gif">
{insert name="getCurrentTime" assign="current_time"
script="time_functions.inc.php"}
The current time is: {$current_time}
```

- The `{literal}` block function, which tells Smarty to ignore anything found inside the tags, `{literal}` and `{/literal}`.

  We need to modify `header.tpl` like this to use the `{literal}` block function:

```
<html>
<head>
 <title>Smarty LLC</title>
</head>
<body>
{literal}
<script language="JavaScript">
<!--
function printMessage(str)
{
    alert(str);
}
//-->
</script>
{/literal}
<img src="images/new.gif">
{insert name="getCurrentTime" assign="current_time"
script="time_functions.inc.php"}
The current time is: {$current_time}
```

Changing the template using either of the options above will fix the Smarty parsing error, and our website should start working again.

Processing Deeply Nested Arrays

While the `{foreach}` function is perfect for handling PHP associative arrays, the `{section}` function is aimed at processing deeply nested arrays, or normal indexed arrays. Let's expand our page that describes the management team of our company by passing a deeply nested array describing each of the team members. This can be done by following these steps:

1. Change the existing `index.php` to the following:

```
<?php
include_once('libs/Smarty.class.php');
$smarty = new Smarty;

if (empty($_GET['language'])) {
    $smarty->assign('language', 'english');
} else {
```


For More Information: www.packtpub.com/smarty/book

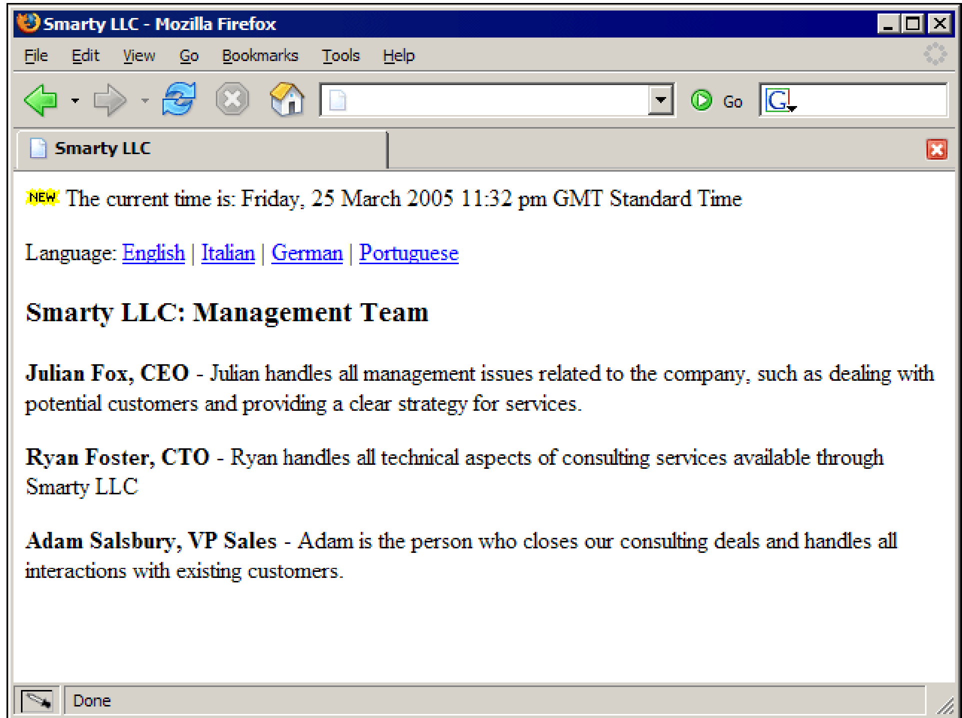
```
    if ($_GET['language'] == 'en') {
        $smarty->assign('language', 'english');
    } elseif ($_GET['language'] == 'it') {
        $smarty->assign('language', 'italian');
    }
}

$team = array(
    0 => array(
            'name'   => 'Julian Fox', // crazy like a fox!
            'title'  => 'CEO',
           'description' => 'Julian handles all management issues related to
                             the company, such as dealing with potential
                             customers and providing a clear strategy
                             for services.',
        ),
    1 => array(
            'name'   => 'Ryan Foster',
            'title'  => 'CTO',
            'description' => 'Ryan handles all technical aspects of
                             consulting services available through
                             Smarty LLC',
        ),
    2 => array(
            'name'   => 'Adam Salsbury', // he will sell you anything!
            'title'  => 'VP Sales',
            'description' => 'Adam is the person who closes our consulting
                             deals and handles all interactions with
                             existing customers.',
        ),
);
$smarty->assign('team', $team);

// list of available languages
$languages = array(
    'en' => 'English',
    'it' => 'Italian',
    'de' => 'German',
    'pt' => 'Portuguese'
);
$smarty->assign('languages', $languages);

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

2. And change the `about.tpl` template file to the following:

```
{config_load file="$language.conf" section="About"}
{include file="header.tpl"}
{include file="navigation.tpl" page_title="Management Team"}

{section name="i" loop=$team}
<p><b>{$team[i].name}, {$team[i].title}</b> - {$team[i].description}</p>
{/section}

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


For More Information: www.packtpub.com/smarty/book

3. Reload the management team page and see the results:



Voilà! You are now dynamically displaying the list of employees based on whatever you pass from the PHP script. The `{section}` function is similar to PHP's `for` construct, in that it loops through an array, and you can treat each of the elements in the array as an associative array. In our example above, we are looping through the list of employees, and using each key in the associative array to display the details of the employee, such as name, title, and description of what they do in the company.

You may also utilize most of the attributes available for the `{foreach}` function, such as `last` (as exemplified before) and more:

- `start`: This allows you to start iterating the given array from the specified position (the first element is zero).
- `step`: This allows you to set a custom step value, and skip certain values. For instance, if you set `step` to `3`, the loop will process records 0, 3, 6 and so on.
- `max`: The maximum number of times that this loop will iterate.
- `show`: Whether to display the output of this function or not.
- `index`: Displays the current iteration, starting from zero.
- `index_prev`: Displays the previous iteration index, if any. It is set to `-1` on the first iteration of the loop.

For More Information: www.packtpub.com/smarty/book

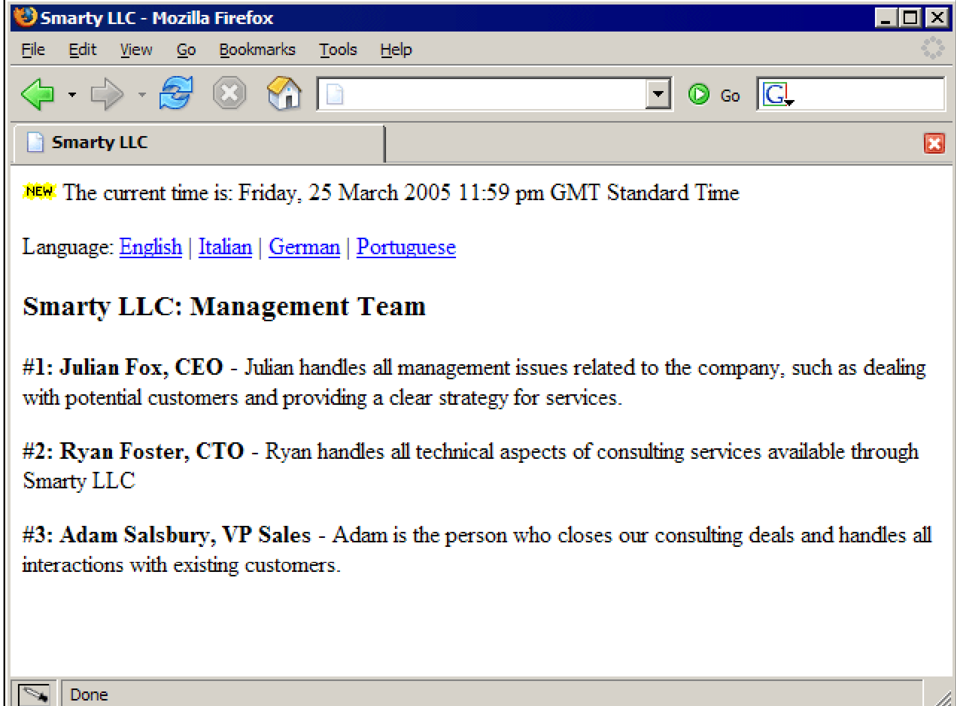
- `index_next`: Displays the next iteration index.
- `iteration`: Displays the current iteration index, starting from one.
- `first`: Set to true if the current iteration is the first one of the loop.
- `last`: Set to true if the current iteration is the last one of the loop.

However, instead of using `$smarty.foreach.foreach_name.last`, you will need to use `$smarty.section.section_name.last`, and so on. In order to better exemplify their usage, let's modify our management team template to display a number before the name of each employee.

Update the `about.tpl` file to contain the following:

```
{config_load file="$language.conf" section="About"}
{include file="header.tpl"}
{include file="navigation.tpl" page_title="Management Team"}
{section name="i" loop=$team}
<p><b>#{$smarty.section.i.iteration}: {$team[i].name}, {$team[i].title}</b> -
{$team[i].description}</p>
{/section}
{include file="footer.tpl"}
```

Reload the page in your browser, and you should see something like this:



For More Information: www.packtpub.com/smarty/book

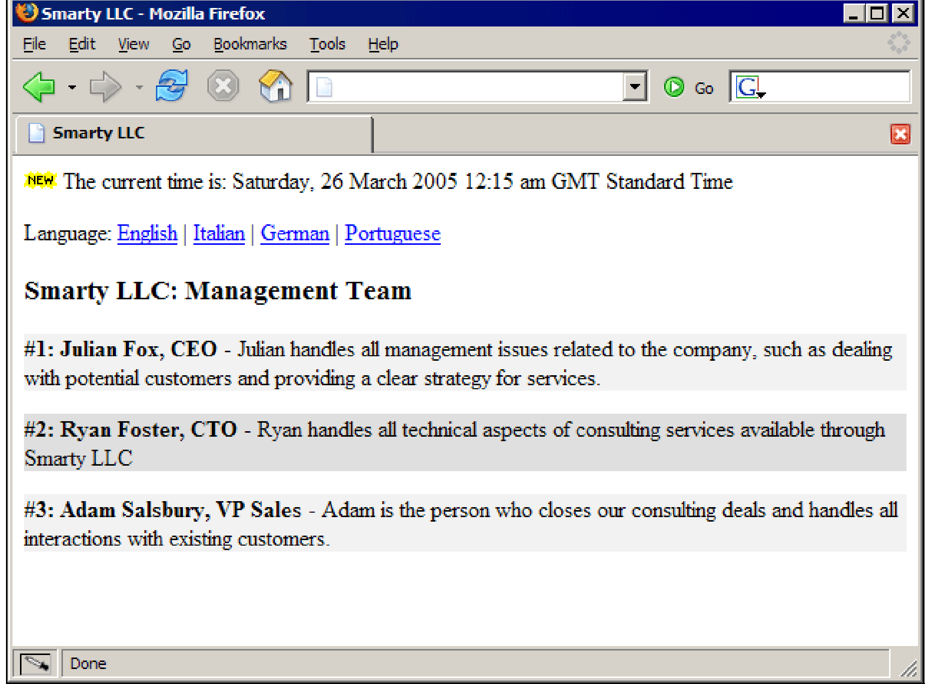
Cycling Through a List of Values

Another very convenient feature in Smarty is the ability to cycle through a pre-defined list of values with the `{cycle}` function. We are going to modify the management team template again, and display employees with alternating background colors.

Change your `about.tpl` file to include the following content:

```
{config_load file="$language.conf" section="About"}
{include file="header.tpl"}
{include file="navigation.tpl" page_title="Management Team"}
{section name="i" loop=$team}
{cycle assign="background_color" values="#F0F0F0,#DDDDDD"}
<p style="background: {$background_color};"><b>#{$smarty.section.i.iteration}:
{$team[i].name}, {$team[i].title}</b> - {$team[i].description}</p>
{/section}
{include file="footer.tpl"}
```

Reload the management team page and you should see the following:



As you can see above, each call to `{cycle}` assigned a color to the `$background_color` variable by cycling through the available options, which should be separated by commas. In our example above, we used only two different colors (or values), but you could use as many as you want, and `{cycle}` will go through them all.

This is especially useful when you have a list of values such as search results to display, and want to make it easy for users to differentiate between one row of results and the next.

For More Information: www.packtpub.com/smarty/book

Avoiding Spam Indexers

A big problem nowadays on the Web is the epidemic of spam, ranging from pop-under advertisements to email-based messages that try to sell you anything from weight-loss solutions to stock tips. However, something that a lot of people don't realize is that just as Google crawls your website to add your content to its index, spam crawlers also look on your site for possible email addresses to send their junk to.

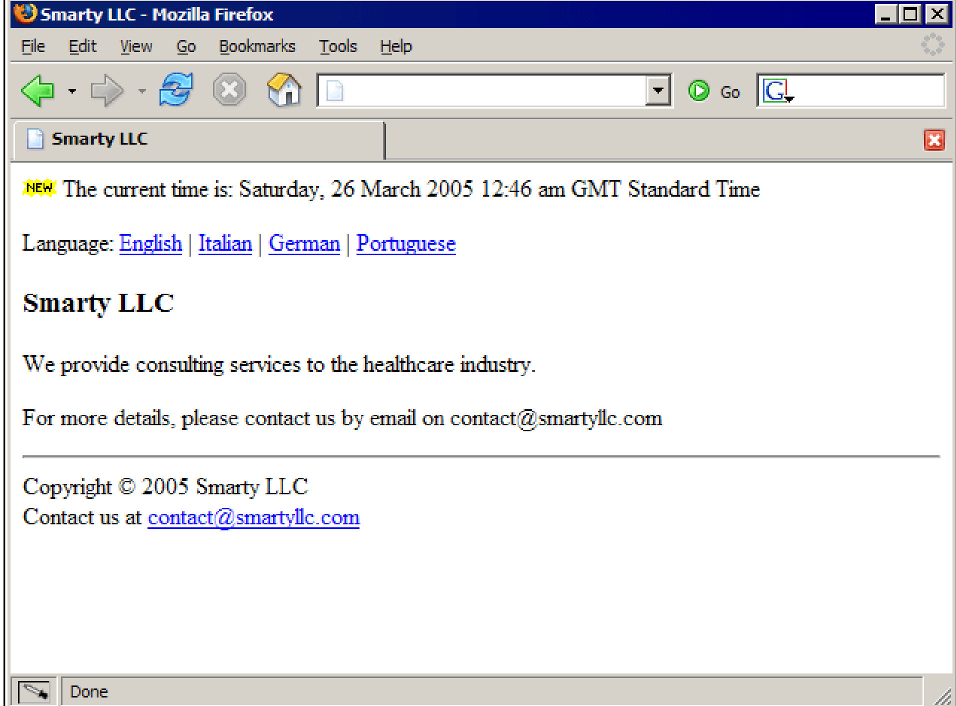
There are pretty reasonable business needs that dictate whether you need to display a contact email address on your website, and if you do, Smarty is there to help you with the `{mailto}` function.

This function allows you to encode an email address in such a way that spam crawlers cannot understand it as a real email address, but normal web browsers can.

Change the `footer.tpl` template file to the following:

```
<hr>
Copyright &copy; 2005 Smarty LLC<br />
Contact us at {mailto address="contact@smartyllc.com" subject="Smarty LLC
Contact" encode="javascript"}

</body>
</html>
```

Reload any of the pages on our website, and you should now see a new section of text on the bottom of the screen, as shown:



For More Information: www.packtpub.com/smarty/book

As you can see, it is simply displaying the email address to the browser. However, the real feature here is what is on the HTML code. The `footer.tpl` portion of HTML got changed to the following after Smarty processed it:

```
<hr>
Copyright &copy; 2005 Smarty LLC<br />
Contact us at <script type="text/javascript"
language="javascript">eval(unescape('%64%6f%63%75%6d%65%6e%74%2e%77%72%69%74%6
5%28%27%3c%61%20%68%72%65%66%3d%22%6d%61%69%6c%74%6f%3a%63%6f%6e%74%61%63%74%4
0%73%6d%61%72%74%79%6c%6c%63%2e%63%6f%6d%3f%73%75%62%6a%65%63%74%3d%53%6d%61%7
2%74%79%25%32%30%4c%4c%43%25%32%30%43%6f%6e%74%61%63%74%22%20%3e%63%6f%6e%74%6
1%63%74%40%73%6d%61%72%74%79%6c%6c%63%2e%63%6f%6d%3c%2f%61%3e%27%29%3b'))</scr
ipt>

</body>
</html>
```

The above will work just fine with browsers that support JavaScript, and spam crawlers will not be able to differentiate this block of JavaScript code from the rest of the HTML.

Form-Related Functions

We will now go over some of the other convenient functions that are very useful when handling form-related data, such as radio boxes, checkboxes, drop-down lists, and so on.

Let's start by creating a PHP script and a template file to handle a page that will allow our prospective customers to request a consulting quote. We will need to gather information such as their name, telephone number and the day, and a time for an initial appointment.

Here's the new `request.php` file:

```
<?php
include_once('libs/Smarty.class.php');
$smarty = new Smarty;

if (empty($_GET['language'])) {
    $smarty->assign('language', 'english');
} else {
    if ($_GET['language'] == 'en') {
        $smarty->assign('language', 'english');
    } elseif ($_GET['language'] == 'it') {
        $smarty->assign('language', 'italian');
    }
}

// list of available languages
$languages = array(
    'en' => 'English',
    'it' => 'Italian',
    'de' => 'German',
    'pt' => 'Portuguese'
);
$smarty->assign('languages', $languages);

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

And this is the template for this new page, to be named `request.tpl`:

```
{config_load file="$language.conf" section="Request"}
{include file="header.tpl"}
{include file="navigation.tpl" page_title="Request a Consulting Quote"}
```


For More Information: www.packtpub.com/smarty/book

```
<form method="post" action="request_quote.php">
<table border="1">
  <tr>
    <td bgcolor="{cycle values="#F0F0F0,#DDDDDD"}">Name:</td>
    <td><input type="text" name="full_name" size="40"></td>
  </tr>
  <tr>
    <td bgcolor="{cycle values="#F0F0F0,#DDDDDD"}">Phone Number:</td>
    <td><input type="text" name="phone" size="20"></td>
  </tr>
  <tr>
    <td bgcolor="{cycle values="#F0F0F0,#DDDDDD"}">
      Appointment Date and Time:
    </td>
    <td>
      {html_select_date display_years=false} 
      {html_select_time display_seconds=false minute_interval=15}
    </td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <input type="submit" value="Request">
    </td>
  </tr>
</table>
</form>

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

It should look something like the following on your browser:

For More Information: www.packtpub.com/smarty/book

This is pretty straightforward. You can see that the call to {html_select_date} will end up displaying two different drop-down boxes, one so the user can select the month for the appointment, and another drop-down box for the day. The call to {html_select_time} also generated two drop-down boxes, one for the hour and the other for the minute.

However, notice that the call to {html_select_date} included a parameter display_years. If that parameter wasn't specified, yet another drop-down box would be displayed, this time listing years. Since our appointment form doesn't need to be that complicated, we set it to false.

Something similar also happened for the call to {html_select_time}. We passed a parameter display_seconds with the value of false to stop Smarty from displaying a drop-down box to select the seconds. Also, we set minute_interval to 15, so the options on the drop-down box end up being 00, 15, 30, and 45.

More Form-Related Functions

Let's expand our Request a Consulting Quote screen to ask prospective customers a few other questions, such as the type of consulting engagement, and payment option.

Change the request.php script to the following:

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

if (empty($_GET['language'])) {
    $smarty->assign('language', 'english');
} else {
    if ($_GET['language'] == 'en') {
        $smarty->assign('language', 'english');
    } elseif ($_GET['language'] == 'it') {
        $smarty->assign('language', 'italian');
    }
}

// list of available consulting types
$smarty->assign('types', array(
    'custom'    => 'Custom Functions',
    'review'    => 'Code Review',
    'database'  => 'Database Design',
));

// list of possible payment options
$smarty->assign('payment', array(
    'pre'     => 'Pre-paid',
    'aswego'  => 'Pay as we go',
));

// list of available languages
$languages = array(
    'en'  => 'English',
    'it'  => 'Italian',
    'de'  => 'German',
    'pt'  => 'Portuguese'
);
$smarty->assign('languages', $languages);

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

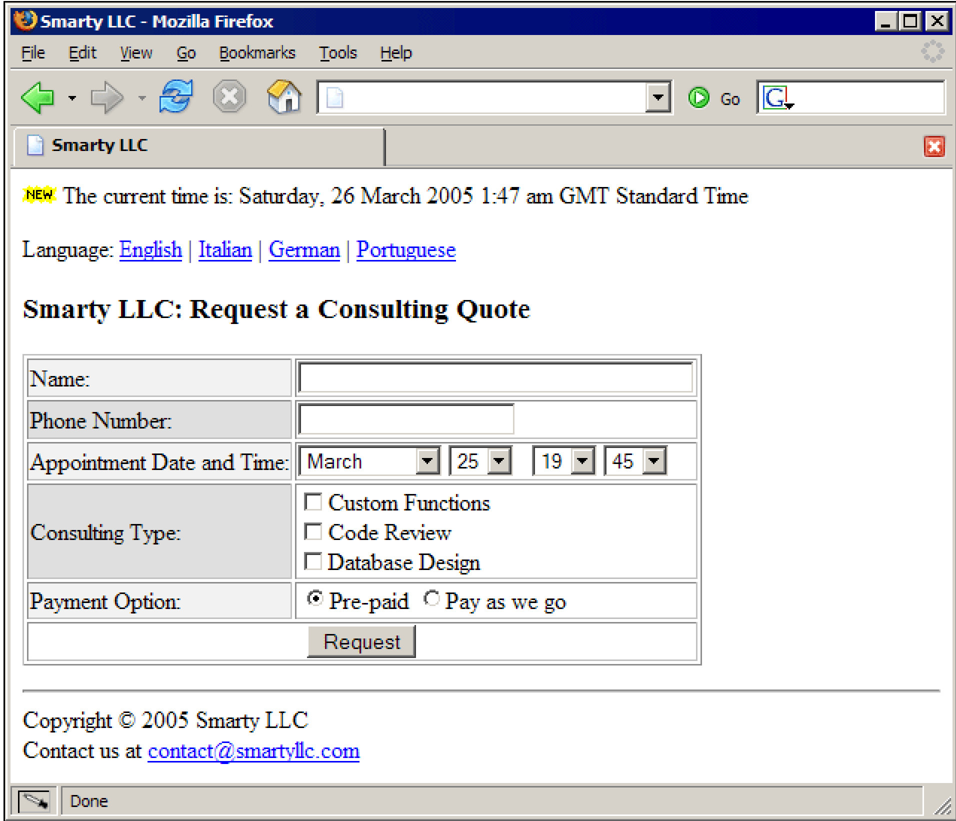

For More Information: www.packtpub.com/smarty/book

Modify the `request.tpl` template file so it contains the following:

```
{config_load file="$language.conf" section="Request"}
{include file="header.tpl"}
{include file="navigation.tpl" page_title="Request a Consulting Quote"}

<form method="post" action="request_quote.php">
<table border="1">
  <tr>
    <td bgcolor="{cycle values="#F0F0F0,#DDDDDD"}">Name:</td>
    <td><input type="text" name="full_name" size="40"></td>
  </tr>
  <tr>
    <td bgcolor="{cycle values="#F0F0F0,#DDDDDD"}">Phone Number:</td>
    <td><input type="text" name="phone" size="20"></td>
  </tr>
  <tr>
    <td bgcolor="{cycle values="#F0F0F0,#DDDDDD"}">
      Appointment Date and Time:
    </td>
    <td>
      {html_select_date display_years=false} 
      {html_select_time display_seconds=false minute_interval=15}
    </td>
  </tr>
  <tr>
    <td bgcolor="{cycle values="#F0F0F0,#DDDDDD"}">
      Consulting Type:
    </td>
    <td>{html_checkboxes name="type" options=$types separator="<br />"}</td>
  </tr>
  <tr>
    <td bgcolor="{cycle values="#F0F0F0,#DDDDDD"}">
      Payment Option:
    </td>
    <td>{html_radios name="payment" options=$payment selected="pre"}</td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <input type="submit" value="Request">
    </td>
  </tr>
</table>
</form>

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


For More Information: www.packtpub.com/smarty/book

Reload the page in your web browser, and you should see something like the following:



There are several interesting features in this template:

- You can build a custom list of checkboxes just by passing a PHP array to the `{html_checkboxes}` function with the `options` parameter, and you can customize how the checkboxes are separated. We are using line breaks here to make it easier for our users to visualize the options of consulting types.
- You can also build a custom list of radio boxes in much the same way, and you can also set the radio box that should be selected by default by using the `selected` parameter to the `{html_radios}` function.

For More Information: www.packtpub.com/smarty/book

Summary

We covered the most important Smarty functions in this chapter, such as `{section}`, `{foreach}`, `{literal}`, `{strip}`, and `{cycle}`, while building a somewhat complex website by starting from a simple set of pages with not much content, and then gradually adding more information and features to our fantasy Smarty LLC company website.

You should now be able to use most Smarty functions comfortably in your own templates, by following the examples available throughout this chapter. Please try to remember that no matter what you need to do in your templates, there is probably somebody else who also needed to do the same, so a built-in Smarty function or even a plug-in might be available to do what you want.

Be sure to always check the Smarty documentation or the appropriate mailing lists. Smarty was designed from the ground up to be easy for template designers like yourself to use, so there's a big probability that what you need to do is already covered in the documentation. Also, there are lots of good people that help out on Smarty-related mailing lists, so make sure to give it a try.

For More Information: www.packtpub.com/smarty/book

Smarty: PHP Template Programming and Applications

Smarty is a powerful templating tool that can breathe new life into PHP programming, and solve many of the difficulties PHP programmers face on non-trivial projects.

This book will take you through the Smarty basics step by step, showing you how to realize the benefits from this product. But this is no "hello world" tutorial; the book also covers advanced Smarty topics crucial to large-scale web development: performance, internationalization, customization, and so on.

Smarty is an established engine with a proven track record for making development and design easier and more elegant. Whether you're just starting with Smarty, or are looking for extra insights on advanced features, this book will help you.

What This Book Covers

The first two chapters are an introduction to Smarty. The remaining chapters of the book are divided into two main sections. Chapters 3-7 deal with Smarty template design, and are aimed primarily at designers. The later chapters get more technical, showing how programmers can work with Smarty and PHP to create complex, interactive and high performance applications. Here's what each chapter covers:

Chapter 1 will ease you into the world of Smarty. Starting with an overview of what templating systems are and why you use them, it goes on to consider Smarty specifically. Once that's covered, you'll see how to install Smarty so you're ready to begin your Smarty development.

Chapter 2 discusses how Smarty can make projects easier for programmers and designers, reduce the time taken on programming and maintenance, and bring harmony to the fraught relationship between designers and programmers. You'll then see how to start your first n-tier Smarty project.

Chapter 3 is a designer's overview of Smarty, explaining the key concepts and how designers can create versatile and attractive templates and layouts.

Chapter 4 looks at templates in detail, showing how values pass into templates and how to use them in applications such as calendars, database-backed reports, email newsletters and more.

Chapter 5 covers more advanced ways to work with templates, including how to play with variables right in the template using modifiers.

Chapter 6 delves into the programming side of Smarty. There are many powerful functions that designers can use to make Smarty development easier and more productive. This chapter introduces the most useful of these, and provides practical examples of functions in action.

With all the added power in the designer's hands, there are bound to be mistakes occasionally.

For More Information: www.packtpub.com/smarty/book

Chapter 7 will introduce you to methods for debugging your Smarty templates, so that you'll be able to find why the wrong values are being displayed, or that pesky extra `<td>` tag is ruining your layout.

Chapter 8 starts with more technical aspects of Smarty. You will see how built-in variables and methods bring added functionality to your PHP code, with very little extra work.

Chapter 9 looks at how Smarty can make the code itself faster. This in-depth look at Smarty's caching and performance features, and how to optimize your code for them, will enable you to write very scalable and fast applications with Smarty.

Chapter 10 pushes Smarty even further, showing how to extend its capabilities with downloadable plug-ins, and even showing how to write your own.

Chapter 11 takes a detailed look at filters, which are a special kind of plug-in.

Chapter 12 concludes the book with a look at how to make your website available in different locales and languages using Smarty's internationalization features.

Where to buy this book

You can buy *Smarty: PHP Template Programming and Applications* from the Packt Publishing website: `http://www.packtpub.com/smarty/book`.

Free shipping to the US, UK, Europe, Australia, New Zealand and India.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information: www.packtpub.com/smarty/book