3
Working with Theme Engines

In this chapter, we will explore theme engines in general and the default PHPTemplate theme engine in detail.

Our exploration of the PHPTemplate engine lays an important foundation for understanding how to create themes or how to extensively modify existing themes. In the examples below, we show the key files used in the process, and how they impact themes. We also discuss the order of precedence among theme files, and how this principle allows us to override the default template files inside individual themes.

We will also discuss the availability of alternatives to the PHPTemplate engine.

Though you don't need to be fluent in PHP to understand this chapter fully, a little familiarity with the programming language will certainly make things easier. The code examples in this chapter come from the Drupal core and the additional themes Gagarin (installed in Chapter 2) and Zen.

What is PHPTemplate?

PHPTemplate is one of a family of applications known as templating engines (referred to frequently in Drupal—and in this text—as "theme engines"). These applications serve a middleware function and determine the coding syntax, which will be used to create the theme. As the name implies, PHPTemplate supports the popular PHP programming language for theme creation.

PHPTemplate was built by developer Adrian Rossouw, and was created specifically for use with Drupal. PHPTemplate is the most widely supported theme engine for Drupal and is compatible with Drupal 4.6 and up. PHPTemplate is included in the default distro of the Drupal 5 series.

Your default PHPTemplate engine files are located on your server in the directory `themes/engines/phptemplate`; additional theme files will appear in the theme directory of each individual PHPTemplate-enabled theme.

PHPTemplate files follow a naming convention: `xyz.tpl.php`.

For example: `block.tpl.php`, `comment.tpl.php`, `node.tpl.php`, `page.tpl.php`

How does it Work?

PHPTemplate is a tool that helps separate the tasks of the programmer from the tasks of the designer. As a tool, PHPTemplate makes it possible for web programmers to work on the business logic of an installation without having to worry too much about the presentation of the content. In contrast, web designers can focus entirely on the styling of discreet blocks of content and items, comprising the layout and the interface. Developers and designers can divide their tasks and optimize their work.

By comparison, other approaches to theming exhibit less flexibility. Themes can be created only with the use of PHP. Pure PHP themes, however, are difficult for those less fluent in the PHP programming language. Pure PHP templates are also hard to read, more difficult to code, and awkward to preview.

Building themes with a theme engine represents a more manageable way of handling dynamic web applications. Every PHPTemplate theme file contains an HTML skeleton with some simple PHP statements for the dynamic data. The theme files are linked together with the CSS files, allowing the dynamic data to be styled and formatted with ease. In other words, PHPTemplate takes one big step towards the oft-heard holy grail of separating the presentation from the content.

The logic included in a PHPTemplate file is generally rather basic, relying primarily on the use of if statements and includes. Much of the code you will see is even more basic and relates purely to the formatting—CSS styling and basic HTML.

The files contained in the PHPTemplate directory on the server (`themes/engines/phptemplate`) work in conjunction with the files located in the active theme's directory (principally the `page.tpl.php` file) to produce the resulting output. The `page.tpl.php` file is the only PHPTemplate file required to enable a theme to employ the theming engine; likewise, all PHPTemplate themes will have this file inside the theme's directory.

Template files are written in PHP and contain a series of includes and conditional statements designed to detect the presence of elements that must be added into the final output. The includes and conditional statements relate to things like the content of the site title, the presence and location of a logo file, the number of active regions, boxes, etc. Whether a statement is satisfied, and the content displayed, is often the product of decisions made by the site administrator in the process of configuring the site as well as decisions made during the creation of content and functionality.

For example, the segment of code below shows the head of a basic `page.tpl.php` file.

```
<head>
   <title>
       98-*
       <?php print $head_title; ?>
   </title>
   <?php print $head; ?>
   <?php print $styles; ?>
   <?php print $scripts; ?>
</head>
```

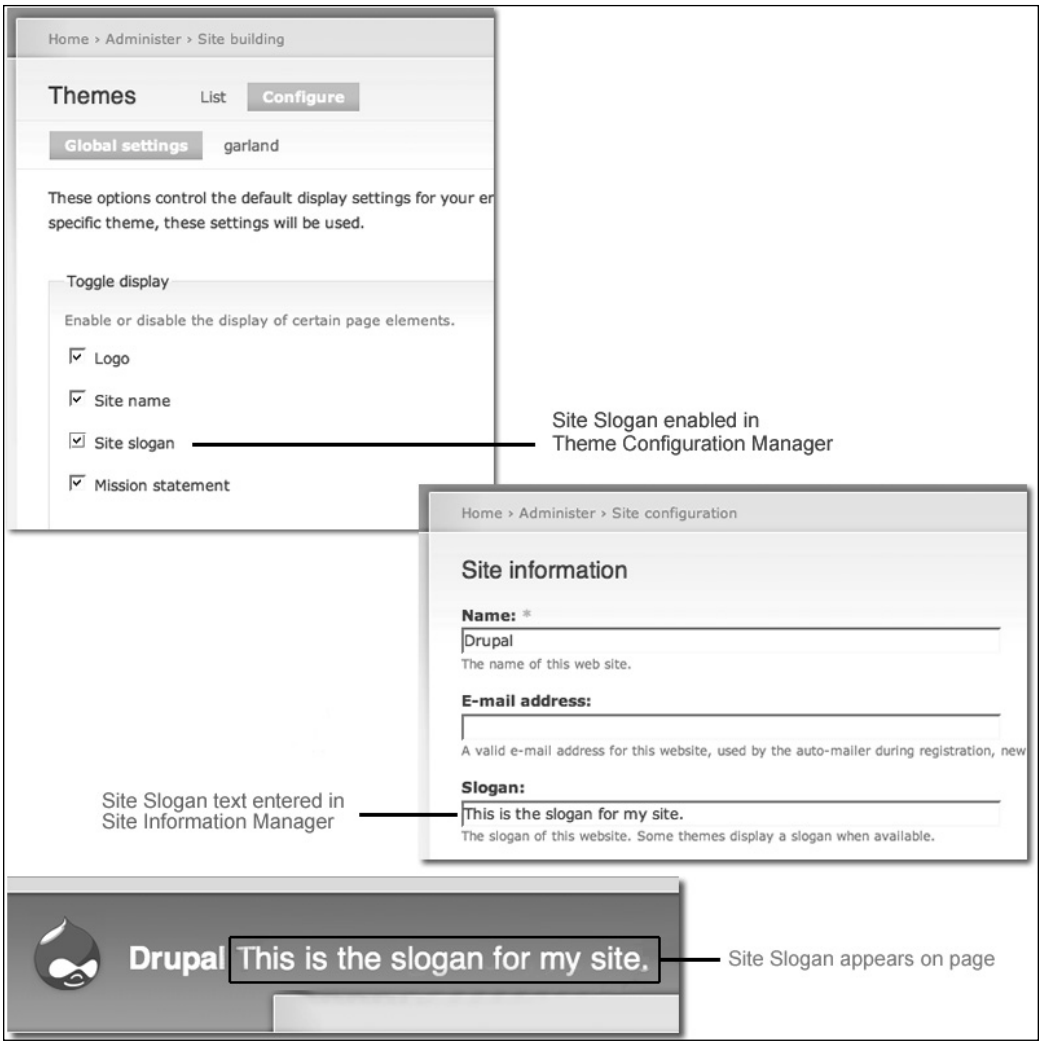
The highlighted lines, above, show the include statements in action; in this case, calling into the template file a variety of information including: the page title (`$head_title`), the head information (`$head`), the style sheets (`$styles`), and any necessary scripts (`$scripts`).

The example below shows a typical application of a conditional statement, again from inside the `page.tpl.php` file:

```
<?php if ($site_slogan): ?>
    <div id="site-slogan">
      <?php print $site_slogan; ?>
    </div>
<?php endif; ?>
```

In this segment, you see a conditional statement testing whether the `$site_slogan` returns as true (i.e., it exists) and if so, it prints the site slogan (`$site_slogan`). You will also note that the site slogan is wrapped with a `div` with an `id` of `site-slogan`. This is our first taste of how CSS integrates with the templates to control the presentation on the screen.

Whether the site slogan is displayed is determined by a parameter specified by the administrator in the Theme Configuration Manager (discussed in Chapter 2). The slogan text is set by the administrator in the site information manager. This parameter's value is stored in the database of your Drupal site.



The choices made by the administrator are stored in the database as `$site_slogan` with the value: `This is the slogan for my site`. `$site_slogan` is then displayed courtesy of a conditional statement in the `page.tpl.php` file.

Putting all this together, it works like this:

1. The `page.tpl.php` looks in the database for the string named `$site_slogan`.
2. If there is a value for `$site_slogan`, `page.tpl.php` then prints that value on the screen.
3. The user's browser applies to the resulting site slogan, the styling specified by the `div` with the `id "site-slogan"`.

The `div` styling in this case is located in the file `style.css`, which is also included in the specific theme's directory. Note also that `style.css` is present courtesy of the actions of the PHPTemplate. The style sheets are included via the statement:

```
<?php print $styles; ?>
```

which appears in the head of the `page.tpl.php` file, as was shown in the previous example.

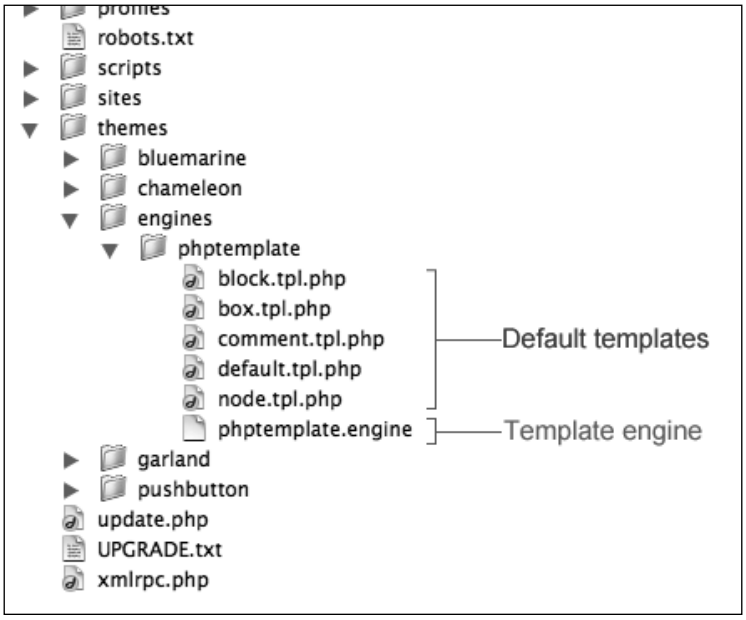
In summary, a complete Drupal theme consists of a number of template files that are combined at run time to present a coherent single web page. The exact number of templates involved and the nature of their contents will vary from theme to theme.

Getting Started with PHPTemplate

Let's take a look at all the key files involved in a PHPTemplate theme. We will start with the default theme engine files, then look at the key file that unites a specific theme to the PHPTemplate theme engine. To illustrate the principles, we will then look at how two different themes approach their implementation with PHPTemplate.

A Look at the Theme Engine Files

Inside the PHPTemplate directory on the server (`themes/engines/phptemplate`), you will find the following:



The default template files contained within the PHPTemplate directory provide the most basic level of formatting, necessary for the styling of various page elements. Here's a brief overview of each of the files contained inside the theme engine directory, along with a short summary of their key functionality:

block.tpl.php

```
<div id="block-<?php print $block->module .'-'. $block->delta; ?>"
class="block block-<?php print $block->module ?>">

<?php if ($block->subject): ?>
  <h2><?php print $block->subject ?></h2>
<?php endif;?>

  <div class="content"><?php print $block->content ?></div>
</div>
```

This template file is used to style the block presentation on the site. Note that the key elements here are the placement of the block subject (note, this is the block's title) and the block's content. The other statements in this file are simply formatting.

box.tpl.php

```
<div class="box">
<?php if ($title): ?>
  <h2><?php print $title ?></h2>
<?php endif; ?>
  <div class="content"><?php print $content ?></div>
</div>
```

This file sets up the wrapping of the content with a "box"—that is, a `div` tag that allows you to format the content along with a title for the box. Note the key elements here are the display of the box's title and the content, each wrapped by styles.

comment.tpl.php

```
<div class="comment<?php print ($comment->new) ? ' comment-new'
: ''; print ($comment->status == COMMENT_NOT_PUBLISHED) ? ' comment-
unpublished' : ''; ?> clear-block">
  <?php print $picture ?>
<?php if ($comment->new) : ?>
  <a id="new"></a>
  <span class="new"><?php print $new ?></span>
<?php endif; ?>
  <h3><?php print $title ?></h3>
  <div class="submitted">
    <?php print $submitted ?>
  </div>
  <div class="content">
    <?php print $content ?>
  </div>
  <?php print $links ?>
</div>
```

This file sets up the display of user-submitted comments to posts and to the forum. Note that the multiple `print` statements here control the display of all aspects of the comment content, including the user's picture, if this option is selected.

default.tpl.php

```
<!-- PHPTemplate was instructed to override the  <?php print $hook ?>
theme function, but no valid template file was found. -->
```

This file is a fallback—a safety net. In situations where a function lacks a valid template, this file is called.

node.tpl.php

```
<div id="node-<?php print $node->nid; ?>" class="node<?php if
($sticky) { print ' sticky'; } ?><?php if (!$status) { print ' node-
unpublished'; } ?> clear-block">

<?php print $picture ?>

<?php if ($page == 0): ?>
  <h2><a href="<?php print $node_url ?>" title="<?php print $title
?>"><?php print $title ?></a></h2>
<?php endif; ?>

  <div class="meta">
  <?php if ($submitted): ?>
    <span class="submitted"><?php print $submitted ?></span>
  <?php endif; ?>

  <?php if ($terms): ?>
    <span class="terms"><?php print $terms ?></span>
  <?php endif;?>
  </div>

  <div class="content">
    <?php print $content ?>
  </div>

<?php
  if ($links) {
    print $links;
  }
?>

</div>
```

Any time a node is rendered, this file is used. This file is the most complicated of the theme files in this directory, and that is because it does a lot of the heavy lifting on the site; this one file works with all the nodes in their many forms.

`block`, `box`, `comment`, and `node` (discussed above) are only the basic default functions. There are, however, many additional functions that can be styled using PHPTemplate. A list of themeable functions and their application is included in the Chapter 4.

template.engine

It's an understatement to say that a lot goes on in this file; a review of the source code of this file will go a long way towards helping you gain an understanding of the big picture of how PHPTemplate assembles the output. Unfortunately, a complete dissertation on the inner workings of PHPTemplate is beyond the scope of this book. Accordingly, I have only highlighted two sections that are of particular interest to anyone who wants to understand how to work with themes.

The first highlighted section enables the regions for use in the theme.

```
/**
 * Declare the available regions implemented by this engine.
 *
 * @return
 *  An array of regions. The first array element will be used as the
default region for themes.
 */
function phptemplate_regions() {
  return array(
       'left' => t('left sidebar'),
       'right' => t('right sidebar'),
       'content' => t('content'),
       'header' => t('header'),
       'footer' => t('footer')
  );
}
```

Note that the above section is perhaps the only place where I will ever endorse directly modifying any file contained in the theme engine directory. You may wish to modify this file if you wish to add or re-name a region across multiple themes; in any other circumstance, I strongly recommend that you stay completely away from making changes to these files. If you need to override these files, do so by creating alternative versions of them that are placed inside the theme directory, alongside the `page.tpl.php` file. This topic is discussed at length in later chapters dealing with intercepts and overrides.

Note the `'t'` function in the above excerpt. This function is related to the translation function, which allows Drupal to show the name for the region in the chosen language inside the administration interface.

The second highlighted section is informational. In this excerpt, the order of precedence among template files is defined. The comments in the code here are very useful; note the example showing how the system will respond to a theme file along each element of the path:

```
// Build a list of suggested template files in order of specificity.
One
  // suggestion is made for every element of the current path, though
  // numeric elements are not carried to subsequent suggestions. For
example,
  // http://www.example.com/node/1/edit would result in the following
  // suggestions:
  //
  // page-node-edit.tpl.php
  // page-node-1.tpl.php
  // page-node.tpl.php
  // page.tpl.php
  $i = 0;
  $suggestion = 'page';
  $suggestions = array($suggestion);
  while ($arg = arg($i++)) {
    $suggestions[] = $suggestion . '-' . $arg;
    if (!is_numeric($arg)) {
      $suggestion .= '-' . $arg;
    }
  }
  if (drupal_is_front_page()) {
    $suggestions[] = 'page-front';
  }

  return _phptemplate_callback('page', $variables, $suggestions);
}
```

The mechanism provided in the example sets out an important principle that is, the order of precedence in the event of the presence of multiple template files. This hierarchy makes it possible for a developer, like you, to create specific templates for specific elements. The option to create themes that can be associated with every element on the path creates a great deal of PHPTemplate's flexibility. Learning to take advantage of that flexibility is one of the key goals of this book.

A Look at the Key PHPTemplate File Contained in the Theme

The template files contained inside the `themes/engines/phptemplate` directory are all linked to another file, `page.tpl.php`, which is located inside the individual theme directory. This file is key to enabling PHPTemplate within a theme.

Some themes use only the basic `page.tpl.php` file to achieve the look and functions the developer desires, others contain a wide variety of additional template files that serve to style specific content or screen space.

For this example, I am using the `page.tpl.php` file from the theme Zen. Zen is not only a representative example of a typical `page.tpl.php` file, but also a particularly useful example due to good use of comments within the code.

```
<!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" lang="<?php print $language
?>" xml:lang="<?php print $language ?>">
<head>
  <title><?php print $head_title; ?></title>
  <?php print $head; ?>
  <?php print $styles; ?>
  <?php print $scripts; ?>
</head>
<?php /* different ids allow for separate theming of the home page */
?>
<body class="<?php print $body_classes; ?>">
  <div id="page">
    <div id="header">
      <div id="logo-title">
        <?php print $search_box; ?>
        <?php if ($logo): ?>
          <a href="<?php print $base_path;
                              ?>" title="<?php print t('Home'); ?>">
            <img src="<?php print $logo;
                      ?>" alt="<?php print t('Home'); ?>" id="logo" />
          </a>
        <?php endif; ?>
        <div id="name-and-slogan">
        <?php if ($site_name): ?>
          <h1 id='site-name'>
            <a href="<?php print $base_path ?>"
                           title="<?php print t('Home'); ?>">
              <?php print $site_name; ?>
            </a>
```

```
            </h1>
          <?php endif; ?>
          <?php if ($site_slogan): ?>
            <div id='site-slogan'>
              <?php print $site_slogan; ?>
            </div>
          <?php endif; ?>
          </div> <!-- /name-and-slogan -->
        </div> <!-- /logo-title -->
        <div id="navigation" class="menu <?php if ($primary_links)
             { print "withprimary"; } if ($secondary_links)
             { print " withsecondary"; } ?> ">
            <?php if ($primary_links): ?>
              <div id="primary" class="clear-block">
                <?php print theme('menu_links', $primary_links); ?>
              </div>
            <?php endif; ?>
            <?php if ($secondary_links): ?>
              <div id="secondary" class="clear-block">
                <?php print theme('menu_links', $secondary_links); ?>
              </div>
            <?php endif; ?>
        </div> <!-- /navigation -->
        <?php if ($header || $breadcrumb): ?>
          <div id="header-region">
            <?php print $breadcrumb; ?>
            <?php print $header; ?>
          </div>
        <?php endif; ?>

      </div> <!-- /header -->
      <div id="container" class="clear-block">
        <?php if ($sidebar_left): ?>
          <div id="sidebar-left" class="column sidebar">
            <?php print $sidebar_left; ?>
          </div> <!-- /sidebar-left -->
        <?php endif; ?>
        <div id="main" class="column"><div id="squeeze">
          <?php if ($mission): ?><div id="mission"><?php print $mission;
                                                  ?></div><?php endif; ?>
          <?php if ($content_top):?><div id="content-top"><?php print
  $content_top; ?></div><?php endif; ?>
```

```
        <?php if ($title): ?><h1 class="title"><?php print $title;
                                                ?></h1><?php endif; ?>
        <?php if ($tabs): ?><div class="tabs"><?php print $tabs;
                                               ?></div><?php endif; ?>
        <?php print $help; ?>
        <?php print $messages; ?>
        <?php print $content; ?>
        <?php print $feed_icons; ?>
        <?php if ($content_bottom): ?><div id="content-bottom"><
                  ?php print $content_bottom; ?></div><?php endif; ?>
      </div></div> <!-- /squeeze /main -->
      <?php if ($sidebar_right): ?>
        <div id="sidebar-right" class="column sidebar">
          <?php print $sidebar_right; ?>
        </div> <!-- /sidebar-right -->
      <?php endif; ?>
    </div> <!-- /container -->
    <div id="footer-wrapper">
      <div id="footer">
        <?php print $footer_message; ?>
      </div> <!-- /footer -->
    </div> <!-- /footer-wrapper -->
    <?php print $closure; ?>
  </div> <!-- /page -->
</body>
</html>
```

You can download your own copy of the Zen theme from `http://drupal.org/project/zen`.

Let's break down this template file, and look at it in bite-sized functional units (we'll leave the CSS until next chapter):

The following code creates the head of the resulting page. The PHP statements in this excerpt include in the resulting web page: the page title, the various bits of head data including the metadata, the style sheets, and the scripts:

```
<head>
  <title><?php print $head_title; ?></title>
<?php print $head; ?>
<?php print $styles; ?>
<?php print $scripts; ?>
</head>
```

This next excerpt begins just inside the beginning of the body of the page The PHP statements here are all conditional—they will only produce output visible to the viewer when the conditions are true. This section includes the optional items controlled by the site administrator, such as the search box, the logo, the site name, and the site slogan. If the administrator has not enabled any of these items, they will not be displayed on the page:

```
<div id="logo-title">
        <?php print $search_box; ?>
        <?php if ($logo): ?>
          <a href="<?php print $base_path;
                              ?>" title="<?php print t('Home'); ?>">
            <img src="<?php print $logo;
                      ?>" alt="<?php print t('Home'); ?>" id="logo" />
          </a>
        <?php endif; ?>

        <div id="name-and-slogan">

        <?php if ($site_name): ?>
          <h1 id='site-name'>
            <a href="<?php print $base_path ?>"
                                  title="<?php print t('Home'); ?>">
              <?php print $site_name; ?>
            </a>
          </h1>
        <?php endif; ?>

        <?php if ($site_slogan): ?>
          <div id='site-slogan'>
            <?php print $site_slogan; ?>
          </div>
        <?php endif; ?>

        </div> <!-- /name-and-slogan -->

      </div> <!-- /logo-title -->
```

This excerpt shows this theme's handling of the navigation:

```
<div id="navigation" class="menu <?php if ($primary_links) { print
"withprimary"; } if ($secondary_links) { print " withsecondary"; } ?>
">
```

The following lines relate to the display of the primary links:

```
          <?php if ($primary_links): ?>
            <div id="primary" class="clear-block">
              <?php print theme('menu_links', $primary_links); ?>
            </div>
          <?php endif; ?>
```

The next segment deals with the secondary links:

```
          <?php if ($secondary_links): ?>
            <div id="secondary" class="clear-block">
              <?php print theme('menu_links', $secondary_links); ?>
            </div>
          <?php endif; ?>
        </div> <!-- /navigation -->
```

This excerpt shows the display of the breadcrumb trail. It also shows the first of this theme's regions, in this case, the header region. In this theme, the header region is declared and active, enabling the site administrator to assign blocks to the region:

```
<?php if ($header || $breadcrumb): ?>
        <div id="header-region">
          <?php print $breadcrumb; ?>
          <?php print $header; ?>
        </div>
      <?php endif; ?>
```

Note that activating a region has two pre-requisites: it must be placed in the `page.tpl.php` file, and the region must also be declared in the `template.engine` file. Adding additional regions to a theme is discussed in detail in later chapters.

This short statement places the left sidebar region on the page. As this theme uses a conditional statement to place this left-hand column on the page, the column will neatly collapse and disappear from view if nothing is assigned to the space:

```
<?php if ($sidebar_left): ?>
        <div id="sidebar-left" class="column sidebar">
          <?php print $sidebar_left; ?>
        </div> <!-- /sidebar-left -->
      <?php endif; ?>
```

This busy excerpt shows a number of events, all of which are associated with the presentation of content items. The statements relate the display of information and functionality with the main content area of the theme:

```
<div id="main" class="column"><div id="squeeze">
```

First is a conditional statement that will display the mission statement (if there is one and it has been enabled by the site administrator):

```
          <?php if ($mission): ?><div id="mission"><
                            ?php print $mission; ?></div><?php endif; ?>
```

The next line places the `content top` region on the page:

```
<?php if ($content_top):?><div id="content-top"><?php print $content_
top; ?></div><?php endif; ?>
```

Next, comes the Item's title:

```
          <?php if ($title): ?><h1 class="title"><?php print $title;
                                                  ?></h1><?php endif; ?>
```

then, the Tabs:

```
<?php if ($tabs): ?><div class="tabs"><?php print $tabs;
                                                ?></div><?php endif; ?>
```

Next, comes the Help link:

```
<?php print $help; ?>
          <?php print $messages; ?>
```

The next line places the `content` region on the page:

```
          <?php print $content; ?>
```

This places the feed icons:

```
          <?php print $feed_icons; ?>
```

The next segment inserts the `content bottom` region on the page. This region, and the `content top` region appear often in themes. The `content top` and `content bottom` regions are typically used by the Drupal system in the layout of certain content items; these regions are not generally available for assignment of blocks:

```
          <?php if ($content_bottom): ?><div id="content-bottom"><
                     ?php print $content_bottom; ?></div><?php endif; ?>
        </div></div> <!-- /squeeze /main -->
```

In later chapters, we will look at how to enable these regions and make them eligible for block assignment.

This excerpt places the right sidebar region on the page. As this theme uses a conditional statement to place this right-hand column on the page, the column will neatly collapse and disappear from view if nothing is assigned to the space:

```
<?php if ($sidebar_right): ?>
        <div id="sidebar-right" class="column sidebar">
          <?php print $sidebar_right; ?>
        </div> <!-- /sidebar-right -->
      <?php endif; ?>
```

This excerpt places the footer region on the page, and also the footer message, if the administrator has included one:

```
<div id="footer-wrapper">
      <div id="footer">
        <?php print $footer_message; ?>
      </div> <!-- /footer -->
    </div> <!-- /footer-wrapper -->
```

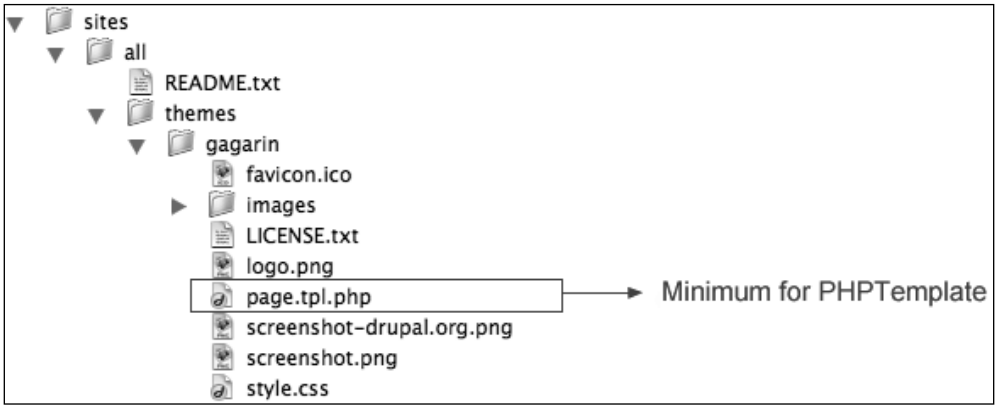
Two Contrasting Examples

As you can probably see, PHPTemplate presents a number of options that can be used to support the creation of themes. You can almost literally do as much or as little as you like.

A look at the range of techniques used by the themes in the market shows a wide variety of approaches to theming. Some themes, like the Gagarin theme we installed in Chapter 2, take a very elemental approach and implement only the bare minimum. Other themes, like the default theme Garland, are more complex, and include optional elements.

A Basic PHPTemplate Theme—Gagarin

The Gagarin theme, shown in the following screenshot, in contrast to Garland, shows the most direct and basic approach to the creation of a PHPTemplate theme. If you check the `sites/all/themes/gagarin` directory on the server, you will find the following files:



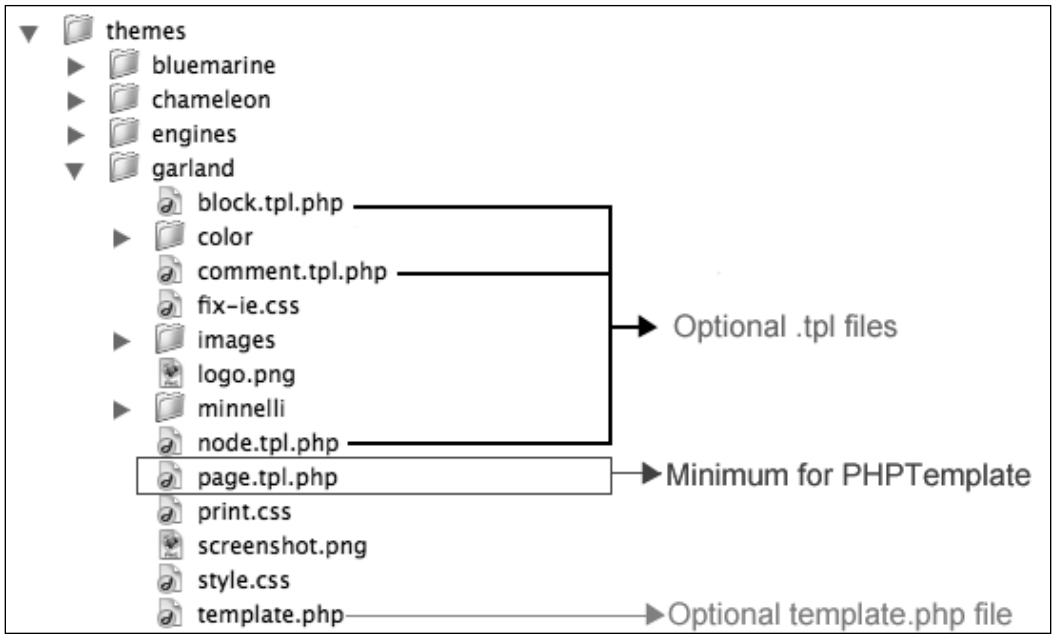
Notice that the creator of Gagarin has chosen to create his theme using the minimum of interaction with the theme engine files. He has used only the `page.tpl.php` file—the bare minimum for enabling PHPTemplate within a theme. He has created no files that intercept or override the default theme files contained in the theme engine directory. Accordingly, the default template files located in the theme engine directory will be used to style the various elements, blocks, boxes, comments, and nodes.

Themes like Gagarin derive their variety from the creative application of CSS.

Themes like Garland derive their variety from both modifying individual theme elements, *and* creative application of CSS.

A More Complex PHPTemplate Theme—Garland

By comparison, Garland shows a more complex approach to the creation of a PHPTemplate theme. If you check the `themes/garland` directory on the server you will find the following files:



Note here that the theme developer has included not only the required `page.tpl.php` file, but has also included his own versions of some `.tpl.php` files, and another additional file, `template.php`.

The files `block.tpl.php`, `comment.tpl.php`, and `node.tpl.php` that are located in the theme's directory are alternative versions of default files included in the theme engine. The system will give precedence to these files over their counterparts in the `themes/engines/phptemplate` directory. This technique—intercepting and overriding the original files—is what allows theme developers to provide extensive alternative styling and layout. Accordingly, the block, comment, and node elements will be handled by the alternative files in the theme directory, while the box element is still governed by the default theme engine file.

Taking the principle one step further, this theme also includes the file `template.php`. The purpose of this file is to specify additional overrides beyond the basic functions: `block`, `box`, `comment`, `node`, and `page`.

If you want to override a theme function not included in the basic list (`block`, `box`, `comment`, `node`, and `page`), you need to tell PHPTemplate about it with the `template.php` file.

Alternative Theme Engines

At the time of writing, the release of Drupal 5.x was only briefly past. Developers of the various templating engines were still working to port their applications to Drupal. While the 4.x series sports a number of templating engine options, including the popular Smarty engine and XTemplate, Drupal 5.x users were left with only one alternative to the default theme engine. Engines that are compatible with the 4.x series are not compatible for the 5.x series.

While at this time only one alternative is certified for Drupal 5.x, for purposes of our discussion here, I'll touch on the most popular alternatives to PHPTemplate. Odds are 5.x users won't have to wait for long before the developers of the popular systems below catch up.

PHPTAL

PHPTAL is a PHP implementation of the ZPT system. At the time of writing this text, PHPTAL was the only alternative to PHPTemplate that was compatible with the Drupal 5.x series.

ZPT stands for Zope Page Templates. ZPT is an HTML/XML generation tool created for use in the Zope project (`http://www.zope.org`). ZPT employs TAL (Tag Attribute Language) to create dynamic templates. Visit the Zope site to learn more about the way of the origins of the system, and how it all works.

TAL is attractive for several reasons. TAL statements come from XML attributes in the TAL namespace that allow you to apply TAL to an XML or plain old HTML document and enable it to function as a template. TAL generates pure, valid XHTML and the resulting template files tend to be clean and easier to read than those created with many other templating engines. One of the biggest advantages, however, is that TAL templates can be manipulated using a standard WYSIWYG HTML/XML editor and previewed in your browser, making the design-work on your theme a relatively easier task.

There are several minor drawbacks to PHPTAL. For purists, it is one level of abstraction further away from PHP, and therefore, performs a bit slower than PHPTemplate (though this difference is unlikely to be noticed by anyone and can be overcome by proper caching). Second, installation of PHPTAL requires Pear5 and PHP5 on your server. If you lack either of these, you should explore other alternatives.

Download PHPTAL for Drupal 5.x at `http://drupal.org/project/phptal`. The Drupal extension includes a variety of extras including at least one PHPTAL theme. You can get the most current PHPTAL snapshot, as well as supporting files, from `http://phptal.motion-twin.com`.

Smarty

The Smarty theme engine allows you to create themes using the Smarty syntax. This popular theme engine is widely used and there are a number of pre-existing themes that are based on Smarty.

Smarty is a mature system and there are a variety of resources to help you learn Smarty's syntax and conventions. Though the system implements another scripting language inside the Drupal system (the Smarty tags), it performs very well. Smarty parses the template files at run time and does not re-compile unless the template files change. Smarty also includes a built-in caching system to help you fine tune performance even further. There are also a variety of plug-ins available, which allow you to extend Smarty's feature set.

Smarty users have been working to get a proper port of Smarty to Drupal 5.x, but at the time of writing, all the efforts were in beta state at best. Nonetheless, given the level of interest and effort, it seems likely a Smarty port for Drupal 5.x will appear soon.

Download Smarty for Drupal 4.7.x at `http://drupal.org/project/smarty`. Smarty's homepage and the most current version of the files can be found at `http://smarty.php.net`.

PHP XTemplate

PHP XTemplate was once the default templating engine in Drupal but has fallen by the wayside as development of the application slowed. For many users, XTemplate was a popular system. It separates the HTML from the PHP and makes it easy for designers to work with themes. Also, as it is written in PHP and can handle either PHP4 or PHP5, it tends to perform well with Drupal.

Unfortunately, at this stage, it seems unlikely to be making a comeback in the near future, and those of you who previously enjoyed using this system should consider alternatives. XTemplate is also released under a different license than Drupal, which may present issues for some users.

Download PHP XTemplate for Drupal 4.7.x at `http://drupal.org/project/xtemplate`. You can visit the project's new home page at `http://www.phpxtemplate.org`. Current files can be found on the SourceForge Project: `http://sourceforge.net/projects/xtpl`.

Installing Additional Theme engines

Additional theme engines can be installed easily. After obtaining the theme engine files, access your server and create a new directory inside of `sites/all/themes`. Name the new directory `engines` and place the theme engine directory inside. Your new theme engine should, in other words, exist inside `sites/all/themes/engines`.

Summary

In this chapter, we've looked in depth at the default PHPTemplate theme engine. You should now have an awareness of the key files involved in a PHPTemplate theme and some appreciation of how those files interact. The discussion of the order of precedence among various theme files lays down a fundamental principle. You have seen example of how to override default theme files by placing alternative files inside the theme directories.

In this chapter, we also spoke about alternative theme engines and noted that although the range of choices is now limited, hopefully we will see more options soon.