CHAPTER 8

The Theme System

Changing the HTML or other markup that Drupal produces requires esoteric knowledge of the layers that make up the theme system. The theme system is an elegant architecture that'll keep you from hacking core code, but it does have a learning curve, especially when you're trying to make your Drupal site look different from other Drupal sites. We'll teach you how the theme system works and reveal some of the best practices hiding within the Drupal core. Here's the first one: you don't need to (nor should you) edit the HTML within module files to change the look and feel of your site. By doing that, you've just created your own proprietary content management system, and have thus lost one the biggest advantages of using a community-supported open source software system to begin with. Override, don't change!

Theme System Components

The theme system comprises several levels of abstraction: template languages, theme engines, and themes.

Template Languages and Theme Engines

The theme system is abstracted to work with most templating languages. Smarty, PHPTAL, and XTemplate can all be used to fill template files with dynamic data within Drupal. To use these languages, a wrapper, called a *theme engine*, is needed to interface Drupal with the corresponding template language. You can find theme engines for the big players of templating languages at `http://drupal.org/project/Theme+engines`. You install theme engines by placing the respective theme engine directory inside the engine directory for your site at `sites/`*`sitename`*`/themes/engine`. To have the theme engine accessible to all sites in a multisite setup, place the theme engine directory inside `sites/all/themes/engine` as shown in Figure 8-1.

The Drupal community has created its own theme engine, optimized for Drupal. It's called PHPTemplate, and it relies on PHP to function as the templating language, which removes the intermediary parsing step other template languages usually go through. This is the most widely supported template engine for Drupal and ships with the core distribution. It's located at `themes/engine/phptemplate`, as shown in Figure 8-2.

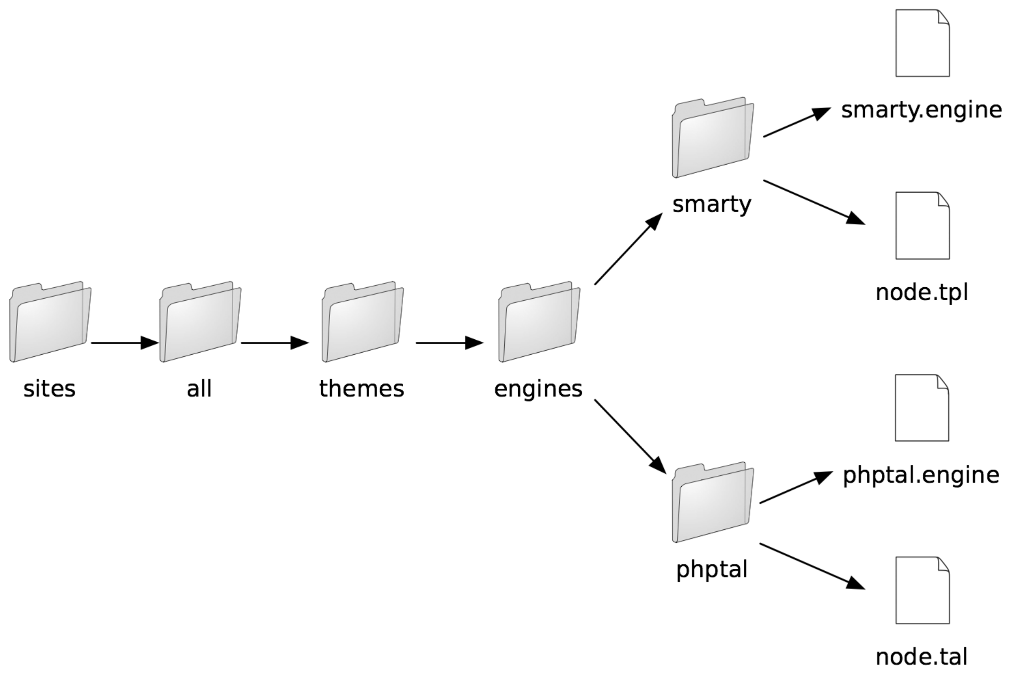


Figure 8-1. *Directory structure for adding custom theme engines to Drupal*

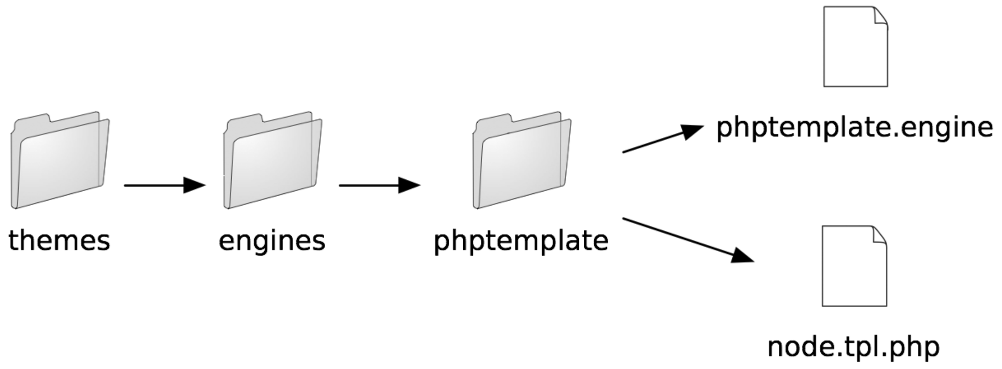


Figure 8-2. *Directory structure for Drupal core theme engines. This location is reserved for core theme engines.*

■**Note** It's entirely possible to skip using a templating language altogether and simply use pure PHP template files. If you're a speed freak or maybe just want to torture your designers, you can even skip using a theme engine and just wrap your entire theme inside PHP functions. For an example of a PHP-based theme, see `themes/chameleon/chameleon.theme`.

Don't expect to see any change to your site after dropping in a new theme engine. Because theme engines are only an interface library, you'll also need to install a Drupal theme that depends on that engine before the theme engine will be used.

Which template language should you use? If you're converting a legacy site, perhaps it's easier to use the previous template language, or maybe your design team is more comfortable working within WYSIWYG editors, in which case PHPTAL is a good choice because it prevents templates from being mangled within those editors. You'll find the most documentation and support for PHPTemplate, and if you're building a new site it's probably your best bet in terms of long-term maintenance and community support.

Themes

In Drupal-speak, themes are a collection of files that make up the look and feel of your site. You can download preconstructed themes from `http://drupal.org/project/Themes`, or you can roll your own, which is what you'll learn to do in this chapter. Themes are made up of most of the things you'd expect to see as a web designer: style sheets, images, JavaScript files, and so on. The difference you'll find between a Drupal theme and a plain HTML site is template files. These files typically contain large sections of HTML and smaller special snippets that are replaced by dynamic content. The syntax of a template file depends on the theme engine that's powering them. For example, take the three snippets of template files in Listings 8-1, 8-2, and 8-3, which output the exact same HTML but contain radically different template file content.

Listing 8-1. *Smarty*

```
<div id="top-nav">
  {if count($secondary_links)}
    <ul id="secondary">
      <li>
        {theme function='links' data=$secondary_links delimiter="</li>\n  <li>"}
      </li>
    </ul>
  {/if}

  {if count($primary_links)}
    <ul id="primary">
      <li>
        {theme function='links' data=$primary_links delimiter="</li>\n    <li>"}
      </li>
    </ul>
  {/if}
</div>
```

Listing 8-2. *PHPTAL*

```
<div id="top-nav">
  <ul tal:condition="php:is_array(secondary_links)" id="secondary">
    <li tal:repeat="link secondary_links" tal:content="link">secondary link</li>
  </ul>

  <ul tal:condition="php:is_array(primary_links)" id="primary">
    <li tal:repeat="link primary_links" tal:content="link">primary link</li>
  </ul>
</div>
```

Listing 8-3. *PHPTemplate*

```
<div id="top-nav">
  <?php if (count($secondary_links)) : ?>
    <ul id="secondary">
    <?php foreach ($secondary_links as $link): ?>
      <li><?php print $link?></li>
    <?php endforeach; ?>
    </ul>
  <?php endif; ?>
  <?php if (count($primary_links)) : ?>
    <ul id="primary">
    <?php foreach ($primary_links as $link): ?>
      <li><?php print $link?></li>
    <?php endforeach; ?>
    </ul>
  <?php endif; ?>
</div>
```

Each template file will look different based on the template language in use. The file extension of a template file is also a dead giveaway to the template language, and thus the theme engine it depends on (see Table 8-1).

Table 8-1. *Template File Extensions Indicate the Template Language They Depend On*

Template File Extension	Theme Engine
.theme	PHP
.tpl.php	PHPTemplate*
.tal	PHPTAL
.tpl	Smarty

** PHPTemplate is Drupal's default theme engine.*

Installing a Theme

To have a new theme show up within the Drupal administrative interface, you should place it in `sites/sitename/themes`. To have the theme accessible to all sites on a multisite setup, place the theme in `sites/all/themes`. You can install as many themes as you want on your site, and themes are installed in much the same way modules are. Once the theme files are in place, navigate over to the administrative interface via Administer ➤ Site building ➤ Themes. You can install and enable multiple themes at once. What does that mean? By enabling multiple themes, users will be able to select one of the enabled themes from within their profile, and that's the theme that will be used when the user browses the site.

When downloading or creating a new theme, it's a best practice to keep the new theme separate from the rest of the core and contributed themes. We recommend creating another level of folders inside your `themes` folder. Place custom themes inside a folder named `custom`, and themes downloaded from the Drupal contributions repository inside a folder named `drupal-contrib`.

Building a PHPTemplate Theme

There are a couple basic ways to create a theme, depending on your starting materials. Suppose your designer has already given you the HTML and CSS for the site. How easy is it to take the designer's design and convert it into a Drupal theme? It's actually not that bad, and you can probably get 80 percent of the way there in short order. The other 20 percent—the final nips and tucks—are what set apart Drupal theming ninjas from lackeys. So let's knock out the easy parts first.

■ **Note** If you're starting your design from scratch, there are many great designs at the Open Source Web Design site at `http://www.oswd.org/`. (Note that these are HTML and CSS designs, not Drupal themes.)

Let's assume you're given the HTML page and style sheet in Listings 8-4 and 8-5 to convert to a Drupal theme. Obviously the files you'd receive in a real project would be more detailed than these, but you get the idea.

Listing 8-4. *page.html*

```
<html>
<head>
  <title>Page Title</title>
  <link rel="stylesheet" href="global.css" type="text/css" />
</head>

<body>
  <div id="container">
    <div id="header">
      <h1>Header</h1>
    </div>
    <div id="sidebar-left">
      <p>
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
        nonummy nibh euismod tincidunt ut.
      </p>
    </div>

    <div id="main">
      <h2>Subheading</h2>
      <p>
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
        nonummy nibh euismod tincidunt ut.
      </p>
    </div>

    <div id="footer">
      Footer
    </div>
  </div>
</body>
</html>
```

Listing 8-5. *global.css*

```
#container {
  width: 90%;
  margin: 10px auto;
  background-color: #fff;
  color: #333;
  border: 1px solid gray;
  line-height: 130%;
}
#header {
  padding: .5em;
  background-color: #ddd;
  border-bottom: 1px solid gray;
}
#header h1 {
  padding: 0;
  margin: 0;
}
#sidebar-left {
  float: left;
  width: 160px;
  margin: 0;
  padding: 1em;
}
```

```
#main {
  margin-left: 200px;
  border-left: 1px solid gray;
  padding: 1em;
  max-width: 36em;
}
#footer {
  clear: both;
  margin: 0;
  padding: .5em;
  color: #333;
  background-color: #ddd;
  border-top: 1px solid gray;
}
#sidebar-left p { margin: 0 0 1em 0; }
#main h2 { margin: 0 0 .5em 0; }
```

The design is shown in Figure 8-3.

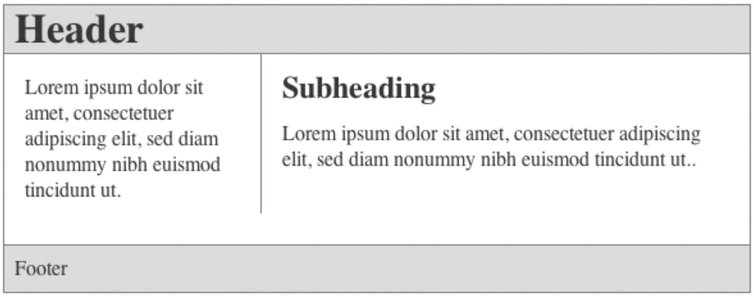

Header
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut.
Subheading
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut..
Footer

Figure 8-3. *Design before it has been converted to a Drupal theme*

Let's call this new theme *greyscale*, so make a folder within `sites/all/themes/custom` called `greyscale`. You might need to create the `themes/custom` folders if you haven't already. Copy `page.html` and `global.css` into the `greyscale` folder. Next, rename `page.html` to `page.tpl.php` so it serves as the new page template for every Drupal page. Because the greyscale theme now has a `page.tpl.php` file, you can enable it within the administrative interface. Go to Administer ➤ Site building ➤ Themes and make it the default theme.

Congratulations! You should now see your design in action. The external style sheet won't yet load (we'll address that later) and any page you navigate to within your site will be the same HTML over and over again, but this is a great start! Any page you navigate to within your site will just serve the static contents of `page.tpl.php`, so . . . there's no way to get to Drupal's administrative interface. We've just locked you out of your Drupal site! Whoops. Getting locked out is bound to happen, and we'll show you now how to recover from this situation. One solution is to rename the folder of the theme currently enabled. In this case you can simply rename `greyscale` to `greyscale_` and you'll be able to get back into the site. That's a quick fix, but because you know what the real problem is, instead you'll add the proper variables to `page.tpl.php` so that the dynamic Drupal content is displayed rather than the static content.

Every PHPTemplate template file—whether page.tpl.php, node.tpl.php, block.tpl.php, and so on—is passed a different set of dynamic content variables to use within the files. Open up page.tpl.php and start replacing the static content with its corresponding Drupal variables. Don't worry, we'll cover what these variables actually do soon.

```
<html>
<head>
  <title><?php print $head_title ?></title>
  <link rel="stylesheet" href="global.css" type="text/css" />
</head>

<body>
  <div id="container">
    <div id="header">
      <h1><?php print $site_name ?></h1>
    </div>

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

    <div id="main">
      <h2><?php print $title ?></h2>
      <?php print $content ?>
    </div>

    <div id="footer">
      <?php print $footer_message ?>
    </div>
  </div>
</body>
</html>
```

Reload your site, and you'll notice that the variables are being replaced with the content from Drupal. Yay! You'll notice that the global.css style sheet isn't loading because the path to the file is no longer correct. You could manually adjust the path, or you could do this the Drupal way and gain some flexibility and benefits.

The first step is to rename global.css to style.css. By convention, Drupal automatically looks for a style.css file for every theme. Once found, it adds this information into the $styles variable that's passed into page.tpl.php. So let's update page.tpl.php with this information:

```
<html>
<head>
  <title><?php print $head_title ?></title>
  <?php print $styles ?>
</head>
...
```

Save your changes and reload the page. Voilà! You'll also notice that if you view the source code of the page, other style sheets from enabled modules have also been added, thanks to the addition of this `$styles` variable. By naming your CSS file `style.css`, you also allow Drupal to apply its CSS preprocessing engine to it to remove all line breaks and spaces from all CSS files, and instead of serving multiple style sheets Drupal will now serve them as a single file. To learn more about this feature, see Chapter 22.

There are plenty more variables to add to `page.tpl.php` and the other template files. So let's dive in!

Understanding Template Files

Some themes have all sorts of template files, while others only have `page.tpl.php`. So how do you know which template files you can create and have be recognized in Drupal? What naming conventions surround the creation of template files? You'll learn the ins and out of working with template files in the following section.

page.tpl.php

`page.tpl.php` is the granddaddy of all template files, and provides the overall page layout for the site. Other template files are inserted into `page.tpl.php`, as the diagram in Figure 8-4 illustrates.

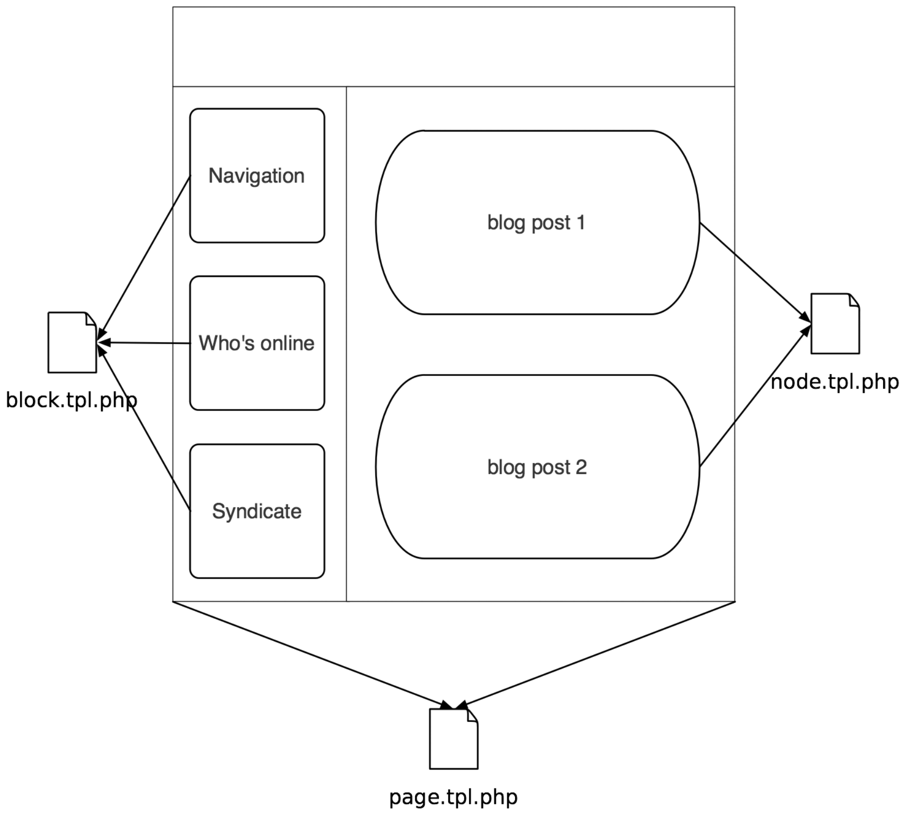


Figure 8-4. *Other templates are inserted within the encompassing page.tpl.php file.*

The insertion of `block.tpl.php` and `node.tpl.php` in Figure 8-4 happens automatically by the theme system. Remember when you created your own `page.tpl.php` file in the previous example? Well, the `$content` variable contained the output of the `node.tpl.php` calls, and `$sidebar_left` contained the output from the `block.tpl.php` calls.

What if you want to create different layouts for other pages on your site, and a single page layout isn't going to cut it? There are two best practices for creating additional page templates.

First, you can create additional page templates within Drupal based on the current system URL of the site. For example, if you were to visit `http://example.com/?q=user/1`, PHPTemplate would look for the following page templates in this order:

```
page-user-1.tpl.php
page-user.tpl.php
page.tpl.php
```

PHPTemplate stops looking for a page template as soon as it finds a template file to include. The `page-user.tpl.php` file would execute for all user pages, whereas `page-user-1.tpl.php` would only execute for the URLs of `user/1`, `user/1/edit`, and so on.

■ **Note** Drupal looks at the internal system URL only, so if you're using the `path` or `pathauto` modules, which allow you to alias URLs, the page templates will still need to reference Drupal's system URL and not the alias.

Let's use the node editing page at `http://example.com/?q=node/1/edit` as an example. Here's the order of template files PHPTemplate would look for:

```
page-node-edit.tpl.php
page-node-1.tpl.php
page-node.tpl.php
page.tpl.php
```

■ **Caution** Even if you don't output the region variables (`$header`, `$footer`, `$sidebar_left`, `$sidebar_right`) within `page.tpl.php`, they are still being built. This is a performance issue because Drupal is doing all that block building only to throw them away for a given page view. If custom page views don't require blocks, a better approach than excluding the variable from the template file is to head over to the block administration interface and disable those blocks from showing on your custom pages. See Chapter 9 for more details on disabling blocks on certain pages.

■ **Tip** To create a custom page template for the front page of your site, simply create a template file named `page-front.tpl.php`.

If you need to make a custom page template, you can start by cloning your current `page.tpl.php`, and then tweak it as needed. The following variables are passed into page templates:

- `$base_path`: The base path of the Drupal installation. At the very least, this will always default to `/` if Drupal is installed in a root directory.
- `$breadcrumb`: Returns the HTML for displaying the navigational breadcrumbs on the page.
- `$closure`: Returns the output of `hook_footer()` and thus is usually displayed at the bottom of the page.
- `$css`: Returns an array structure of all the CSS files to be added to the page. Use `$styles` to return the HTML version of the `$css` array.
- `$content`: Returns the HTML content to be displayed. Examples include a node, an aggregation of nodes, the content of the administrative interface, and so on.
- `$directory`: The relative path to the directory the theme is located in; for example, `themes/bluemarine` or `sites/all/themes/custom/mytheme`. You'll commonly use this variable in conjunction with the `$base_path` variable to build the absolute path to your site's theme:

  ```
  <?php print $base_path . $directory ?>
  ```

- `$feed_icons`: Returns RSS feed links for the page.
- `$footer_message`: Returns the text of the footer message that was entered at Administer ➤ Site configuration ➤ Site information.
- `$head`: Returns the HTML to be placed within the `<head></head>` section. Modules append to `$head` by calling `drupal_set_html_head()` to add additional markup such as RSS feeds.
- `$head_title`: The text to be displayed in the page title, between the HTML `<title></title>` tags.
- `$help`: Help text, mostly for administrative pages. Modules can populate this variable by implementing `hook_help()`.
- `$is_front`: `TRUE` if the front page is currently being displayed.
- `$language`: The language in which the site is being displayed.
- `$layout`: This variable allows you to style different types of layouts, and the value for `$layout` depends on the number of sidebars enabled. Possible values include `none`, `left`, `right`, and `both`.
- `$logo`: The path to the logo image, as defined in the theme configuration page of enabled themes. It's used as follows in Drupal's core themes:

  ```
  <img src="<?php print $logo ?>" />
  ```

- `$messages`: This variable returns the HTML for validation errors and success notices for forms and other messages as well. It's usually displayed at the top of the page.
- `$mission`: Returns the text of the site mission that was entered at Administer ➤ Site configuration ➤ Site information. This variable is only populated when `$is_front` is `TRUE`.

- `$node`: The entire node object, available when viewing a single node page.
- `$primary_links`: An array containing the primary links as they have been defined at Administer ➤ Site building ➤ Menus. Usually `$primary_links` is styled through the `theme('links')` function as follows:

  ```
  <?php print theme('links', $primary_links) ?>
  ```

- `$scripts`: Returns the HTML for adding the `<script>` tags to the page. This is also how jQuery is loaded (see Chapter 17 for more on jQuery).
- `$search_box`: Returns the HTML for the search form. `$search_box` is empty when the administrator has disabled the display on the theme configuration page of enabled themes or if search module is disabled.
- `$secondary_links`: An array containing the secondary links as they have been defined at Administer ➤ Site building ➤ Menus. Usually `$secondary_links` is styled through the `theme('links')` function as follows:

  ```
  <?php print theme('links', $secondary_links) ?>
  ```

- `$sidebar_left`: Returns the HTML for the left sidebar, including the HTML for blocks belonging to this region.
- `$sidebar_right`: Returns the HTML for the right sidebar, including the HTML for blocks belonging to this region.
- `$site_name`: The name of the site, which is set at Administer ➤ Site configuration ➤ Site information. `$site_name` is empty when the administrator has disabled the display on the theme configuration page of enabled themes.
- `$site_slogan`: The slogan of the site, which is set at Administer ➤ Site configuration ➤ Site information. `$site_slogan` is empty when the administrator has disabled the display of the slogan on the theme configuration page of enabled themes.
- `$styles`: Returns the HTML for linking to the necessary CSS files to the page. CSS files are added to the `$styles` variable through `drupal_add_css()`.
- `$tabs`: Returns the HTML for displaying tabs such as the View/Edit tabs for nodes. Tabs are usually at the top of the page in Drupal's core themes.
- `$title`: The main content title, different from `$head_title`. When on a single node view page `$title` is the title of the node. When viewing Drupal's administration pages, `$title` is usually set by the menu item that corresponds to the page being viewed (see Chapter 4 for more on menu items).

node.tpl.php

Node templates are responsible for controlling individual pieces of content displayed within a page. Rather than affecting the entire page, node templates only affect the `$content` variable within `page.tpl.php`. They're responsible for the presentation of nodes in teaser view (when multiple nodes are listed on a single page), and also in body view (when the node fills the entire

`$content` variable in `page.tpl.php` and stands alone on its own page). The `$page` variable within a node template file will be `TRUE` when you're in body view or `FALSE` if you're in teaser view.

The `node.tpl.php` file is the generic template that handles the view of all nodes. What if you want a different template for, say, blogs than forum posts? How can you make node templates for a specific node type rather than just a generic catch-all template file?

The good news is that node templates offer a refreshing level of granularity that's not entirely obvious out of the box. Simply cloning `node.tpl.php` and renaming the new file to `node-`*`nodetype`*`.tpl.php` is enough for PHPTemplate to choose this template over the generic one. So theming blog entries is as simple as creating `node-blog.tpl.php`. Any node type you create via Administer ➤ Content management ➤ Content types can have a corresponding node template file in the same fashion. You can use the following default variables in node templates:

- `$content`: The body of the node, or the teaser if it's a paged result view.
- `$date`: The formatted date the node was created.
- `$links`: The links associated with a node, such as "read more" and "add comment." Modules add additional links by implementing `hook_link()`.
- `$name`: Formatted name of the user who authored the page, linked to his or her profile.
- `$node`: The entire node object and all its properties.
- `$node_url`: The permanent URI to this node.
- `$page`: `TRUE` if the node is being displayed by itself as a page. `FALSE` if it is on a multiple node listing view.
- `$taxonomy`: An array of the node's taxonomy terms.
- `$teaser`: Boolean to determine whether or not the teaser is displayed. This variable can be used to indicate whether `$content` consists of the node body (`FALSE`) or teaser (`TRUE`).
- `$terms`: HTML containing the taxonomy terms associated with this node. Each term is also linked to its own taxonomy term pages.
- `$title`: Title of the node. Will also be a link to the node's body view when on a multiple node listing page.
- `$submitted`: "Submitted by" text. The administrator can configure display of this information in the theme configuration page on a per-node-type basis.
- `$picture`: HTML for the user picture, if pictures are enabled and the user picture is set.

Often the `$content` variable within node template files doesn't structure the data the way you'd like it to. This is especially true when using contributed modules that extend a node's attributes, such as Content Construction Kit (CCK) field-related modules.

Luckily, PHPTemplate passes the entire node object to the node template files. If you write the following debug statement at the top of your node template file and reload a page containing a node, you'll discover all the properties that make up the node. It's probably easier to read if you view the source of the page you browse to.

```
<pre>
  <?php print_r($node) ?>
</pre>
```

Now you can see all the components that make up a node, access their properties directly, and thus mark them up as desired, rather than work with an aggregated `$content` variable.

■ **Caution** When formatting a node object directly, you also become responsible for the security of your site. Please see Chapter 20 to learn how to wrap user-submitted data in the appropriate functions to prevent XSS attacks.

---

block.tpl.php

Blocks are listed on Administer ➤ Site building ➤ Blocks and are wrapped in the markup provided by `block.tpl.php`. If you're not familiar with blocks, please see Chapter 9 for more details. Like the page template and node template files, the block system uses a suggestion hierarchy to find the template file to wrap blocks in. The hierarchy is as follows:

```
block-modulename-delta.tpl.php
block-modulename.tpl.php
block-region.tpl.php
block.tpl.php
```

In the preceding sequence, *`modulename`* is the name of the module that implements the block. For example the "Who's Online" block is implemented by `user.module`. Blocks created by the site administrator are always tied to the block module. If you don't know the module that implemented a given block, you can find all the juicy details by doing some PHP debugging. By typing in the following one-liner at the top of your `block.tpl.php` file, you print out the entire block object for each block that's enabled on the current page:

```
<pre>
  <?php print_r($block); ?>
</pre>
```

This is easier to read if you view the source code of the web browser page. Here's what it looks like for the "Who's Online" block:

```
stdClass Object
(
  [module] => user
  [delta] => 3
  [theme] => bluemarine
  [status] => 1
  [weight] => 0
  [region] => footer
  [custom] => 0
  [throttle] => 0
  [visibility] => 0
  [pages] =>
  [title] =>
  [subject] => Who's online
  [content] => There are currently ...
)
```

Now that you have all the details of this block, you can easily construct one or more of the following block template files, depending on the scope of what you want to target:

```
block-user-3.tpl.php // Target just the Who's Online block.
block-user.tpl.php   // Target all block output by user module.
block-footer.tpl.php // Target all blocks in the footer region.
block.tpl.php        // Target all blocks on any page.
```

Here's a list of the default variables you can access within block template files:

- `$block`: The entire block object.
- `$block_id`: An integer that increments each time a block is generated and the block template file is invoked.
- `$block_zebra`: Whenever `$block_id` is incremented it toggles this variable back and forth between "odd" and "even."

comment.tpl.php

The `comment.tpl.php` template file adds markup to comments. It's not as easy as it is with nodes to tell Drupal to mark up blog comments differently from forum comments. It can be done, but requires programming and delving into the `phptemplate_variables()` function. The following variables are passed into the comment template:

- `$author`: Hyperlink author name to the author's profile page, if he or she has one.
- `$comment`: Comment object containing all comment attributes.
- `$content`: The body of the comment.
- `$date`: Formatted creation date of the post.
- `$links`: Contextual links related to the comment such as "edit, "reply," and "delete."
- `$new`: Returns "new" for a comment yet to be viewed by the currently logged in user and "updated" for an updated comment. You can change the text returned from `$new` by overriding `theme_mark()` in `includes/theme.inc`. Drupal doesn't track which comments have been read or updated for anonymous users.
- `$picture`: HTML for the user picture. You must enable picture support at Administer ➤ User management ➤ User settings, and you must check "User pictures in comments" on each theme's configuration page for enabled themes. Finally, either the site administrator must provide a default picture or the user must upload a picture so there is an image to display.

- `$submitted`: “Submitted by” string with username and date.
- `$title`: Hyperlink title to this comment.

box.tpl.php

The `box.tpl.php` template file is one of the more obscure template files within Drupal. It’s used in Drupal core to wrap the comment submission form and search results. Other than that, it doesn’t have much use. It serves no function for blocks, as one might erroneously think (because blocks created by the administrator are stored in a database table named `boxes`). You have access to the following default variables within the box template:

- `$content`: The content of a box.
- `$region`: The region in which the box should be displayed. Examples include `header`, `sidebar-left`, and `main`.
- `$title`: The title of a box.

Advanced Drupal Theming

In the previous section you learned about the different template files Drupal looks for when it’s putting your theme together. You learned how to swap out page templates and how to create node-type-specific templates and even block-specific template files. In other words, you’ve acquired the knowledge to build out 80 percent of your custom theme.

What about the other 20 percent? How do you theme Drupal’s forms? How do you tweak something as simple as the breadcrumb trail? In this section we’ll answer those questions and teach you how to become a Drupal theming ninja. You’ll start by learning the theming ninja’s weapon of choice: the `template.php` file.

The `template.php` file is the place to wire up custom template files, define new block regions, override Drupal’s default theme functions, and intercept and create custom variables to pass along to template files.

Overriding Theme Functions

The core philosophy behind Drupal’s theme system is similar to that of the hook system. By adhering to a naming convention, functions can identify themselves as theme-related functions that are responsible for formatting and returning your site’s content. Themeable functions are identifiable by their function names, which all begin with `theme_`. This naming convention gives Drupal the ability to create a function-override mechanism for all themeable functions. Designers can instruct Drupal to execute an alternative function which takes precedence over the theme functions that module developers expose. For example, let’s examine how this process works when building the site’s breadcrumb trail.

Open up includes/theme.inc and examine the functions inside that file. Almost every function in there begins with theme_, which is the telltale sign it can be overridden. In particular, let's examine theme_breadcrumb():

```
/**
 * Return a themed breadcrumb trail.
 *
 * @param $breadcrumb
 *   An array containing the breadcrumb links.
 * @return a string containing the breadcrumb output.
 */
function theme_breadcrumb($breadcrumb) {
  if (!empty($breadcrumb)) {
    return '<div class="breadcrumb">'. implode(' È ', $breadcrumb) .'</div>';
  }
}
```

This function controls the HTML for the breadcrumb navigation within Drupal. Currently it adds a right-pointing double-arrow separator between each item of the trail. Suppose you want to change the div tag to a span and use an asterisk (*) instead of a double arrow. How should you go about it? One solution would be to edit this function within theme.inc, save it, and call it good. (No! No! Do *not* do this!) There are better ways.

Have you ever seen how these theme functions are invoked within core? You'll never see theme_breadcrumb() called directly. Instead, it's always wrapped inside the theme() helper function.

You'd expect the function to be called as follows:

```
theme_breadcrumb($breadcrumb)
```

But it's not. Instead, you'll see developers use the following invocation:

```
theme('breadcrumb', $breadcrumb);
```

This generic theme() function is responsible for initializing the theme layer and the dispatching of function calls to the appropriate places, bringing us to the more elegant solution to our problem. The call to theme() instructs Drupal to look for the breadcrumb functions shown in Figure 8-5, in this order.

Assuming the theme you're using is bluemarine, which is a PHPTemplate-based theme, then Drupal would look for the following:

```
bluemarine_breadcrumb()
phptemplate_breadcrumb()
theme_breadcrumb()
```

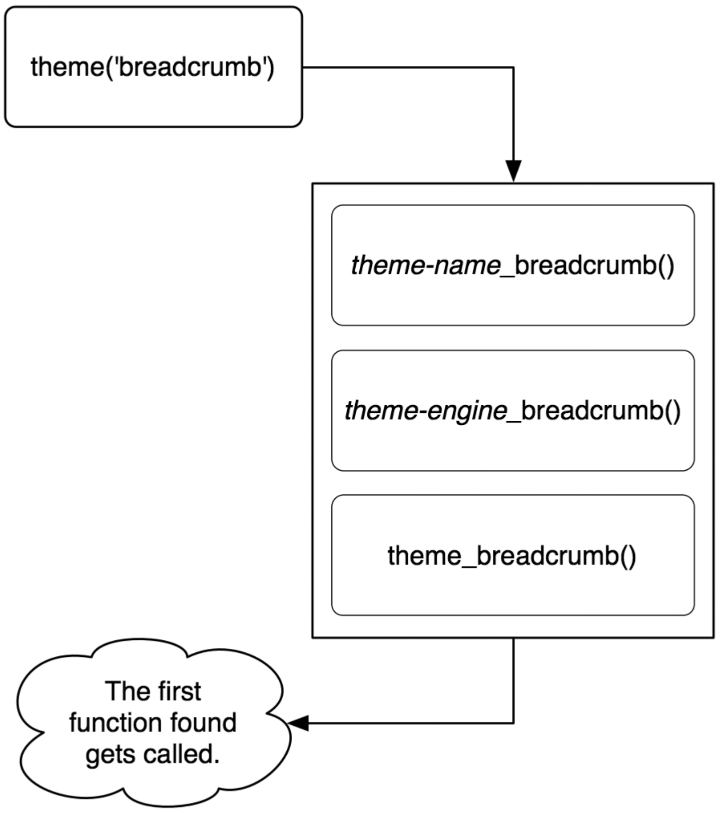
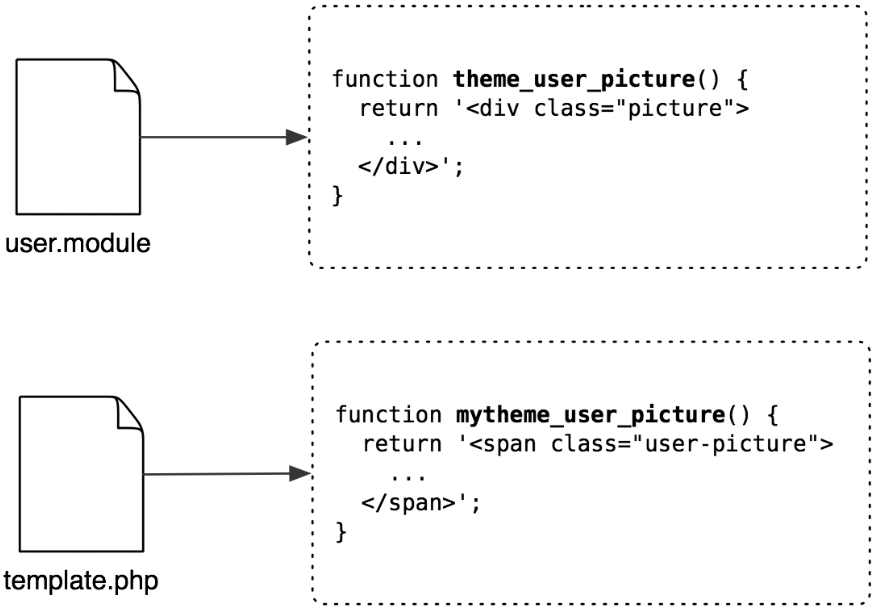

theme('breadcrumb')
theme-name_breadcrumb()
theme-engine_breadcrumb()
theme_breadcrumb()
The first function found gets called.

Figure 8-5. *How theme overriding works*

To tweak the Drupal breadcrumbs, create an empty `template.php` file in your current theme's folder and copy and paste the `theme_breadcrumb()` function in there from `theme.inc`. Be sure to include the starting `<?php` tag. Also, rename the function from `theme_breadcrumb` to *`the-name-of-your-theme`*`_breadcrumb`.

```
<?php
/**
 * Return a themed breadcrumb trail.
 *
 * @param $breadcrumb
 *   An array containing the breadcrumb links.
 * @return a string containing the breadcrumb output.
 */
function mytheme_breadcrumb($breadcrumb) {
  if (!empty($breadcrumb)) {
    return '<span class="breadcrumb">'. implode(' * ', $breadcrumb) .'</span>';
  }
}
```

The next time Drupal is asked to format the breadcrumb trail, it'll find your function first and use it instead of the default `theme_breadcrumb()` function, and breadcrumbs will contain your asterisks instead of Drupal's double arrows. Pretty slick, eh? And you haven't touched a line of core code. By passing all theme function calls through the `theme()` function, Drupal will always check if the current theme has overridden any of the `theme_` functions and call those instead. Developers, take note: any parts of your modules that output HTML or XML should only be done within theme functions so they become accessible for themers to override. Take `user.module`, for example (see Figure 8-6).



```
function theme_user_picture() {
  return '<div class="picture">
    ...
  </div>';
}
```

```
function mytheme_user_picture() {
  return '<span class="user-picture">
    ...
  </span>';
}
```

Figure 8-6. *Modules expose theme-related functions that can be overriden at the theme level. The custom theme function mytheme_user_picture() overrides the default function theme_user_picture() and is called instead.*

Defining Additional Template Files

If you're working with a designer, telling him or her to "just go in the code and find the themeable functions to override" is out of the picture. Fortunately, there's another way to make this more accessible to designer types. You can instead map themeable functions to their own template files. We'll demonstrate with our handy breadcrumb example.

First, create a file within your theme directory named `breadcrumb.tpl.php`. This is the new template file for breadcrumbs. Because we wanted to change the `<div>` tag to a `<span>` tag, go ahead and populate the file with the following:

```
<span class="breadcrumb"><?php print $breadcrumb ?></span>
```

That's easy enough for a designer to edit. Now you need to let Drupal know to call this template file when looking to render its breadcrumbs. Inside `template.php`, override `theme_breadcrumb()` as you did previously, but this time you're going to tell this function to use the template file instead of just the function:

```
function mytheme_breadcrumb($breadcrumb) {
  if (!empty($breadcrumb)) {
    $variables = array(
      'breadcrumb' =>   implode(' * ', $breadcrumb)
    );
    return _phptemplate_callback('breadcrumb', $variables);
  }
}
```

The magic inside this function is happening with `_phptemplate_callback()`. Its first parameter is the name of the template file to look for, and the second parameter is an array of variables to pass to the template file. You can create and pass along as many variables as you need into your template files.

Adding and Manipulating Template Variables

The question becomes: if you can make your own template files and control the variables being sent to them, how can you manipulate or add variables being passed into page and node templates?

Every call to load a template file passes through the `phptemplate_callback()` function to which you were just introduced. This function is responsible for aggregating the variables to pass along to the correct template file, and these variables can come from three different locations:

- The `$variables` array passed in as a second parameter to `_phptemplate_callback()`.
- The default variables appended to every template file via `_phptemplate_default_variables()` in `phptemplate.engine`.
- The variables returned from `_phptemplate_variables()`. This function doesn't exist in Drupal by default, and it's where you can manipulate the variables for every template file within your theme before it's sent to the template file.

Figure 8-7 shows the big picture of how this ties into the larger theme system.

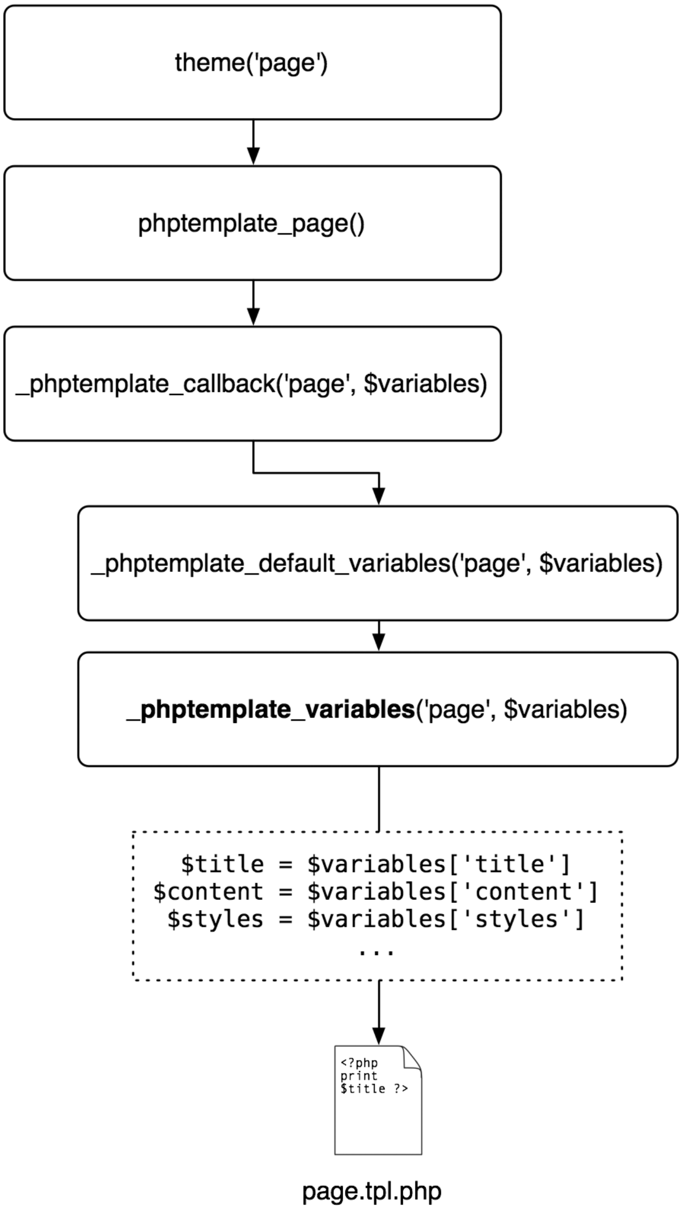

theme('page')
phptemplate_page()
_phptemplate_callback('page', $variables)
_phptemplate_default_variables('page', $variables)
_phptemplate_variables('page', $variables)
$title = $variables['title']
$content = $variables['content']
$styles = $variables['styles']
...
<?php print $title ?>
page.tpl.php

Figure 8-7. *Developers can intercept and manipulate the variables sent to template files through the _phptemplate_variables() function.*

A common usage of _phptemplate_variables() is to set a variable when someone is visiting the site and that person is logged in. Add the following code to your template.php file:

```
/**
 * Intercept template variables.
 *
 * @param $hook
 *   The name of the theme function being executed.
 * @param $vars
 *   An array of variables passed to the template file.
 */
function _phptemplate_variables($hook, $vars = array()) {
  switch ($hook) {
    // Send a new variable, $logged_in, to page.tpl.php to tell us if the current
    // user is logged in or out.
    case 'page':
      // Get the currently logged in user.
      global $user;

      // An anonymous user has a user ID of zero.
      if ($user->uid > 0) {
        // The user is logged in.
        $vars['logged_in'] = TRUE;
      }
      else {
        // The user is not logged in.
        $vars['logged_in'] = FALSE;
      }
      break;
  }

  return $vars;
}
```

In the preceding code, you created a new variable to be passed into the page theme hook, or page.tpl.php for all intents and purposes. You set $logged_in to TRUE when a user is logged in and FALSE when an anonymous user is visiting the site. Another common variable to set is to check when an author is creating a comment on a post he or she has written, so it can be styled differently. Here's how to do that:

```
function _phptemplate_variables($hook, $vars = array()) {
  switch ($hook) {
    // Send a new variable, $logged_in, to page.tpl.php to tell us if the current
    // user is logged in or not.
    case 'page':
      // Get the currently logged in user.
      global $user;
      // An anonymous user has a user id of zero.
      if ($user->uid > 0) {
        // The user is logged in.
        $vars['logged_in'] = TRUE;
      }
      else {
        // The user is not logged in.
        $vars['logged_in'] = FALSE;
      }
      break;

    case 'comment':
      // We load the node object to which the current comment is attached.
      $node = node_load($vars['comment']->nid);
      // If the author of this comment is equal to the author of the node,
      // we set a variable; then in our theme we can theme this comment
      // differently so it stands out.
      $vars['author_comment'] = $vars['comment']->uid == $node->uid ? TRUE
        : FALSE;
      break;
  }

  return $vars;
}
```

Now in `comment.tpl.php` you can check the value of `$author_comment` and set a special CSS class based on its value.

■**Note** One of the variables you can change within `_phptemplate_variables()` is `$vars['template_file']`, which is the name of the template file Drupal is about to call. If you need to load an alternate template file based on a more complex condition, this is the place to do it.

Defining New Block Regions

Regions in Drupal are areas in themes where blocks can be placed. You assign blocks to regions and organize them within the Drupal administrative interface at Administer ➤ Site building ➤ Blocks.

The default regions used in themes are `left sidebar`, `right sidebar`, `header`, and `footer`, although you can create as many regions as you want. Once declared, they're made available to your page template files (for example, `page.tpl.php`) as a variable. For instance, use `<?php print $header ?>` for the placement of the `header` region. You create additional regions by creating a function named *`name-of-your-theme`*`_regions()` within your `template.php` file.

```
/*
 * Declare the available regions implemented by this engine.
 *
 * @return
 *    An array of regions. Each array element takes the format:
 *    variable_name => t('human readable name')
 */
function mytheme_regions() {
  return array(
       'left' => t('left sidebar'),
       'right' => t('right sidebar'),
       'content_top' => t('content top'),
       'content_bottom' => t('content bottom'),
       'header' => t('header'),
       'footer' => t('footer')
  );
}
```

To print out the `content top` region in your page template, use `<?php print $content_top ?>`.

Theming Drupal's Forms

Changing the markup within Drupal forms isn't as easy as creating a template file, because forms within Drupal are dependent on their own API. Chapter 10 covers how to map theme functions to forms in detail.

Summary

After reading this chapter you should be able to

- Understand what theme engines and themes are
- Understand how PHPTemplate works within Drupal
- Create template files
- Override theme functions
- Manipulate template variables
- Create new page regions for blocks