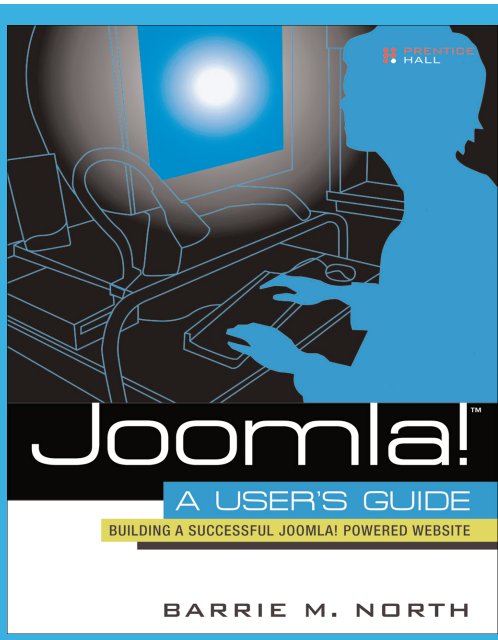

Published Dec 11, 2007 by Prentice Hall.

ISBN-10: 0-13-613560-9
ISBN-13: 978-0-13-613560-9

www.informit.com/ph

Chapter 9

Creating a Pure CSS Template

In this chapter, we'll go through the steps of creating a Joomla template. Specifically, we will create a template that uses Cascading Style Sheets (CSS) to produce a layout without use of tables. This is a desirable goal because it means that the template code is easier to validate to World Wide Web Consortium (W3C) standards. It also tends to load faster, be easier to maintain, and perform better in search engines. These issues are discussed in detail later in the chapter.

In This Chapter

- What is a Joomla template? What functions are performed by a Joomla template, and what is the difference when a template has no content versus when content is added into the Content Management System (CMS).
- How does the localhost design process differ to that of a static (X)HTML web page?
- What are the implications of tableless designs in Joomla and the relationship between W3C standards, usability, and accessibility?
- What files make up a Joomla template, and what functions do they perform?

- How do you create a source-ordered 3-column layout using CSS rather than tables?
- What are the basic CSS styles that should be used with Joomla, and what are the default styles that are used by the Joomla core?
- How do you place and style modules, and what are some new techniques for rounded corners?
- What would be a simple strategy to produce lean CSS menus that mimic the effect of those developed with JavaScript?
- How do you control when columns are shown and hide them when no content is present?
- What are the proper steps to create a real Joomla 1.5 template?

What Is a Joomla Template?

A Joomla template is a series of files within the Joomla CMS that control the presentation of the content. The Joomla template is not a website; it's also not considered a complete website design. The template is the basic foundation design for viewing your Joomla website. To produce the effect of a "complete" website, the template works hand in hand with the content stored in the Joomla databases. An example of this can be seen in Figure 9.1.

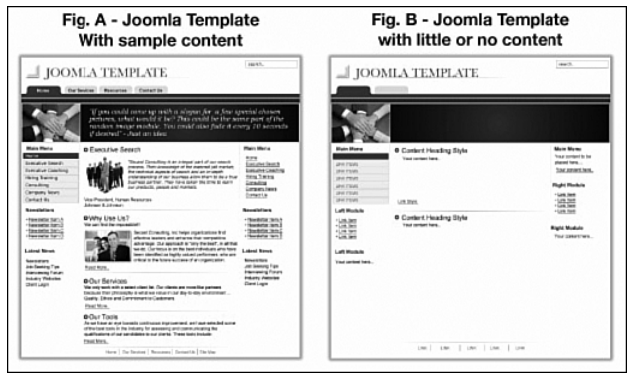

FIGURE 9.1 Template with and without content

Figure 9.1, part A, shows the template in use with sample content. Part B shows the template as it might look with a raw Joomla installation with little or no content. The template is styled so that when your content is inserted, it will inherit the stylesheet defined in the template such as link styles, menus, navigation, text size, and colors to name a few.

Notice that the images associated with the content (the photos of the people) are not part of the template but the header is.

Using a template for a CMS, as Joomla does, has a number of advantages and disadvantages:

- There is a complete separation of content and presentation, especially when CSS is used for layout (as opposed to having tables in the index.php file). This is one of the main criteria for a site that meets modern web standards.
- A new template, and hence a completely new look to a website, can be applied instantly. This can even have different locations/positioning of content as well as colors and graphics.
- If different layouts are called for within one website, it can be difficult to achieve.

Although different templates can be applied to different pages, this built-in functionality is not reliable. Much better is to use conditional PHP and create a layout that dynamically adjusts the number of columns based on what content is published.

The Least You Need to Know
Modern websites separate content from presentation using a technology known as Cascading Style Sheets (CSS). In Joomla, the template controls the presentation of the content.

Localhost Design Process

The web page you see at a Joomla-powered website is not static. That means it is generated dynamically from content stored in the database. The page that you see is created through various PHP commands that are in the template, which presents some difficulties in the design phase.

It's common now to use a What You See Is What You Get (WYSIWYG) HTML editor, such as Dreamweaver. This means that the designer does not even need to code the HTML. However, this is not possible in the Joomla template design process

because WYSIWYG editors cannot display a dynamic page. This means that the designer must code "by hand" and view the output page from the PHP on a served page. With a fast enough connection this could be a web server, but most designers use a "local server" on their own computer. This is a piece of software that will serve the web pages on the designer's computer.

There is no "right way" to create a web page; it depends on the designer's background. Those more graphics-inclined make an "image" of a page in a graphics program like Photoshop and then break up the images to be able to use them for the Web (known as slice and dice). More technology-based designers will often just jump straight into the CSS and start coding. However, as just mentioned, the Joomla template designer is limited in that he cannot instantly see the effect of his coding in the same editor. The modified design process is as follows:

1. Make edits with HTML editor, save changes.
2. Have localhost server running in background to "run" Joomla.
3. View edits in a web browser.
4. Return to step 1.

The Least You Need to Know
When creating a template, you have to have Joomla "running" on a server so you can make changes and refresh the page output.

Localhost Server Options

In Chapter 2, "Downloading and Installing Joomla," we saw how to install a web server that will run on your computer. We described one for a webserver called WAMP5. To move further along in this chapter, you will need to have this installed. If you haven't yet, go ahead and install it. I'll wait right here.

TIP
One useful technique to make the design process more efficient is to serve a web page that you are designing and then copy and paste the source into an editor. For example, once your layout CSS is set up, you can use one of these localhost servers to serve a page, then view the source of the page. You then copy and paste the source code into your editor. You can now easily style the page using CSS and not have to go through the cycle of steps described earlier.

NOTE
A Free XHTML Editor
For those not able to pay for a commercial editor, such as Dreamweaver, some free editors are available. Nvu is a solid choice and has built-in validation—and it is 100% open source. This means anyone is welcome to download Nvu at no charge (nvu.com/download.html), including the source code if you need to make special changes.

W3C and Tableless Design

Usability, accessibility, and search engine optimization (SEO) are all phrases used to describe high-quality web pages on the Internet today. In reality, there is a significant amount of overlap between usability, accessibility and SEO and a web page that demonstrates the characteristics of one does so for all three; this is shown in Figure 9.2. The easiest way to achieve these three goals is to do so using the framework laid out in the World Wide Web Consortium (W3C) web standards.

For example, a site that is structured semantically with (X)HTML (the (X)HTML explains the document, not how it looks) will be easily read through a screen reader by someone who has poor vision. It will also be easily read by a search engine spider. Google is effectively blind in how it reads your website, it's as though it is using a screen reader.

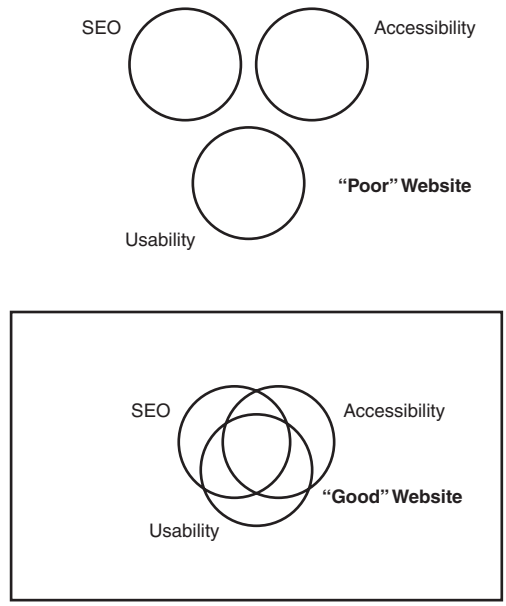

FIGURE 9.2 The overlap between usability, accessibility, and SEO

Web standards put into place a common set of "rules" for all web browsers to use to display a web page. The main organization pushing these standards is the World Wide Web Consortium (W3C), whose Director, Tim Berners-Lee, has the distinction of actually inventing the Web in 1989.

To help you understand where web standards came from, some history is helpful. Many web pages are actually designed for older browsers. Why? Browsers have continually evolved since the World Wide Web started. New ones have appeared, and some old ones have disappeared (remember Netscape?).

Current W3C standards serve to (hopefully) push manufacturers to release more compliant browsers so that designers can design to one common platform.

Another complicating factor is that different browser makers (like Microsoft) tend to have their browsers interpret html/xhtml in slightly different ways. This has lead to web designers having to design their websites to support older browsers rather than new ones. It's often decided that it's important that a web page appear properly to these "legacy" browsers. The W3C standards outlined for web page code have been developed to achieve consistency. A site that incorporates the W3C's web standards has a much better foundation for making itself accessible, usable, and search engine-optimized. Think of these as building codes for your house. A website built with them is stronger and safer and coincides with users' expectations. You can check your pages with the W3C's HTML validation service (validator.w3.org/). It's easy and free (make sure you use the correct DOCTYPE when you try and validate your code[1]). At its simplest, a site that meets W3C validation uses semantic (X)HTML and separates content from presentation using CSS.

Ask five designers what web standards are, and you will get five different answers. But most agree that they are based on using valid code, whether HTML or (X)HTML (or others).

Semantically Correct Code

As mentioned earlier, being semantic means that the (X)HTML in the web page describes only content, not presentation. In particular, this means structured organization of H1,H2 tags etc and only using tables for tabular data, not layout.

Cascading Style Sheets (CSS)

Closely related to having semantic code, is using Cascading Style Sheets (CSS) to control the look and layout of a web page. CSS is a simple mechanism for adding style

(that is, fonts, colors, spacing, and so on) to Web documents (source: www.w3.org/Style/CSS/). They exist parallel to the (X)HTML code and so let you completely separate content (semantic code) from presentation (CSS). The best example of this is CSS Zen Garden, a site where the same semantic (X)HTML is shaped in different and unique ways with different CSS. The result is pages that look very different but have the same core content.

Designing Joomla-powered sites currently presents considerable challenges to meet validation standards. In the first series of releases, 1.0.X, the code used a significant amount of tables to output its pages. This isn't really using CSS for presentation, nor does it produce semantically correct code. This problem is compounded by the fact that very few third-party developers are using CSS; most use tables to generate their code too.

Fortunately, the Joomla Core Development team recognized this issue with Joomla. In the 1.5 version, it's possible for template designers to completely override the output of the core (called a view) and strip out the tables or customize the layout—whatever they want.

Regardless, care can still be taken when creating a template to make sure it is accessible (for example, scalable font sizes), usable (clear navigation) and optimized for search engines (source-ordered).

The Least You Need to Know
Creating valid templates should be a path, not a goal. The idea is to make your template as accessible as possible for humans and spiders, not to achieve a badge of valid markup.

Creating a Simple Template

To understand the contents of a template, we will start by looking at a blank Joomla template.

The Template File Components

The template contains the various files and folders that make up a Joomla template. These files must be placed in the `/templates/` directory of a Joomla installation in their own folder. So if we had two templates installed, our directory would look something like the following:

```
/templates/element
/templates/voodoo
```

Note that the directory names for the templates must be the same as the name of the template, in this case element and voodoo. Obviously they are case sensitive and shouldn't contain spaces.

Within the directory of a template, there are a number of key files:

```
/element/templateDetails.xml
/element/index.php
```

These two filenames and locations must match exactly because this is what they are called by the Joomla core script.

The first of these is the template XML file.

```
templateDetails.xml
```

This is an XML format metadata file that tells Joomla what other files are needed when loading a web page that uses this template. Note the uppercase "D." It also details the author, copyright, and what files make up the template (including any images used). The last use of this file is for installing a template when using the admin backend.

Second, we have the engine of the template, the `index.php`:

```
index.php
```

This file is the most important. It lays out the site and tells the Joomla CMS where to put the different components and modules. It is a combination of PHP and (X)HTML.

In almost all templates, additional files are used. It is conventional (although not required by the core) to name and locate them as shown here:

```
/element/template_thumbnail.png
/element/css/template.css
/element/images/logo.png
```

These are just examples. Table 9.1 examines each line.

TABLE 9.1 Core Files Needed for a Template

/templatename/folder/filename	Description
/element/template_thumbnail.png	A web browser screenshot of the template (usually reduced to around 140 pixels wide and 90 pixels high). After the template has been installed, this functions as a "Preview Image" visible in the Joomla administration Template Manager and also the template selector module in the frontend (if used).

/templatename/folder/filename	Description
/element/css/template.css	The CSS of the template. The folder location is optional, but you have to specify where it is in the `index.php` file. You can call it what you like. Usually the name shown is used, but we will see later that there are advantages in having other CSS files too.
/element/images/logo.png	Any images that go with the template. Again for organization reasons, most designers put this in an images folder. Here we have an image file called `logo.png` as an example.

templateDetails.xml

The `templateDetails.xml` must include all the files that are part of the template. It also includes information such as the author and copyright. Some of these are shown in the admin backend in the Template Manager. An example XML file is shown here:

```
<?xml version="1.0" encoding="utf-8"?>
<install version="1.5" type="template">
      <name>TemplateTutorial15</name>
      <creationDate>August 2007</creationDate>
      <author>Barrie North</author>
      <copyright>GPL</copyright>
      <authorEmail> compassdesigns@gmail.com </authorEmail>
      <authorUrl>www.compassdesigns.net</authorUrl>
      <version>1.0</version>
      <description>First example template for Chapter 9 of the Joomla Book
      </description>
      <files>
            <filename>index.php</filename>
            <filename>templateDetails.xml</filename>
            <filename>js/somejsfile.js</filename>
            <filename>images/threecol-l.gif</filename>
            <filename>images/threecol-r.gif</filename>
            <filename>css/customize.css</filename>
            <filename>css/layout.css</filename>
            <filename>css/template_css.css</filename>
       </files>
      <positions>
            <position>user1</position>
            <position>top</position>
            <position>left</position>
            <position>banner</position>
            <position>right</position>
            <position>footer</position>
      </positions>
```

```
        <params>
           <param name="colorVariation" type="list" default="white" label="Color
Variation" description="Color variation to use">
                 <option value="blue">Blue</option>
                 <option value="red">Red</option>
           </param>
        </params>
</install>
```

Let's explain what some of these lines mean:

- **`<install version="1.5" type="template">`**. The contents of the XML document are instructions for the backend installer. The option `type="template"` tells the installer that we are installing a template and that it is for Joomla 1.5.
- **`<name>TemplateTutorial15</name>`**. Defines the name of your template. The name you enter here will also be used to create the directory within the templates directory. Therefore it should not contain any characters that the file system cannot handle, for example spaces. If installing manually, you need to create a directory that is identical to the template name.
- **`<creationDate>August 2007</creationDate>`**. The date the template was created. It is a free form field and can be anything such as May 2005, 08-June-1978, 01/01/2004, and so on.
- **`<author>Barrie North</author>`**. The name of the author of this template—most likely your name.
- **`<copyright>GPL</copyright>`**. Any copyright information goes into this element. A Licensing Primer for Developers and Designers can be found in the Joomla forums.
- **`<authorEmail>compassdesigns@gmail.com</authorEmail>`**. Email address where the author of this template can be reached.
- **`<authorUrl>www.compassdesigns.net</authorUrl>`**. The URL of the author's website.
- **`<version>1.0</version>`**. The version of this template.
- **`<files></files>`**. Various files used in the template.

  The files used in the template are laid out with `<filename>` tags:

```
        <files>
             <filename>index.php</filename>
             <filename>templateDetails.xml</filename>
```

```
        <filename>js/somejsfile.js</filename>
        <filename>images/threecol-l.gif</filename>
        <filename>images/threecol-r.gif</filename>
        <filename>css/customize.css</filename>
        <filename>css/layout.css</filename>
        <filename>css/template_css.css</filename>
    </files>
```

- The "files" sections contain all generic files like the PHP source for the template or the thumbnail image for the template preview. Each file listed in this section is enclosed by `<filename> </filename>`. Also included would be any additional files; here the example of a JavaScript file that is required by the template is used.
- All image files that the template uses are also listed in the `<files>` section. Again, each file listed is enclosed by `<filename> </filename>`. Path information for the files is relative to the root of the template. For example, if the template is in the directory called 'YourTemplate', and all images are in a directory 'images' that is inside 'YourTemplate', the correct path is: `<filename>images/my_image.jpg</filename>`.
- Last, any stylesheets are listed in the files section. Again, the filename is enclosed by `<filename> </filename>`, and it's path is relative to the template root.
- **`<positions></positions>`**.The module positions available in the template.
- **`<params></params>`**. These describe parameters that can be passed to allow advanced template functions such as changing the color of the template.

index.php

What actually is in an `index.php` file? It is a combination of (X)HTML and PHP that determines everything about the layout and presentation of the pages.

First, let's look at a critical part of achieving valid templates, the DOCTYPE at the top of the `index.php` file. This is the bit of code that goes at the very top of every web page. At the top of our page, we have this in our template:

```
<?php
// no direct access
defined( '_JEXEC' ) or die( 'Restricted access' );
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

The first PHP statement simply makes sure that the file is not accessed directly for security.

A web page DOCTYPE is one of the fundamental components of how a web page is shown by a browser, specifically, how that browser interprets CSS. To give you further understanding, an observation from alistapart.com says

> [Information on W3C's site about DOCTYPEs is] written by geeks for geeks. And when I say geeks, I don't mean ordinary web professionals like you and me. I mean geeks who make the rest of us look like Grandma on the first day of She's Got Mail.

Anyway, you can use several DOCTYPEs. Basically, the DOCTYPE tells the browser how to interpret the page. Here the words "strict" and "transitional" start getting floated around (`float:left` and `float:right` usually). Essentially, ever since the Web started, different browsers have had different levels of support for CSS. This means for example, that Internet Explorer won't understand the "min-width" command to set a minimum page width. To duplicate the effect, you have to use "hacks" in the CSS.

NOTE
Some say that serving (X)HTML as text/html is considered harmful. If you actually understand that statement you are well ahead of the game and beyond this guide. You can read more at hixie.ch/advocacy/xhtml.

Strict means the HTML (or (X)HTML) will be interpreted exactly as dictated by standards. A transitional DOCTYPE means that the page will be allowed a few agreed upon differences to the standards.

To complicate things, there is something called "quirks" mode. If the DOCTYPE is wrong, outdated, or not there, the browser goes into quirks mode. This is an attempt to be backwards-compatible, so Internet Explorer 6 for example, will render the page pretending as if it were IE4.

Unfortunately, people sometimes end up in quirks mode accidentally. It usually happens in two ways:

- They use the DOCTYPE declaration straight from the WC3 web page, and the link ends up as `DTD/xhtml1-strict.dtd`, except this is a relative link on the WC3 server. You need the full path as shown earlier.
- Microsoft set up IE6 so you could have valid pages but be in quirks mode. This happens by having an "xml declaration" put before the DOCTYPE.

Next is an XML statement (after the DOCTYPE):

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php echo $this->language;
 ?>" lang="<?php echo $this->language; ?>" >
```

The part about IE6 quirks mode is important. In this chapter we only design for IE6+, so we will make sure that it's running in standards mode. This will minimize the hacks we have to do later on.

NOTE

Making a page standards-compliant, where you see "valid xhtml" at the bottom of the page does not mean really difficult coding, or hard-to-understand tags. It merely means that the code you use matches the DOCTYPE you said it would. That's it! Nothing else.

Designing your site to standards can, on one level, be reduced to saying what you do and then doing what you say.

Here are some useful links, which will help you understand DOCTYPE and quirks mode:

- www.quirksmode.org/css/quirksmode.html
- www.alistapart.com/stories/doctype
- www.w3.org/QA/2002/04/Web-Quality
- http://forum.joomla.org/index.php/topic,7537.0.html
- http://forum.joomla.org/index.php/topic,6048.0.html

What Else Is in `index.php`?

Let's look at the structure of the header first; we want to be as minimal as possible but still have enough for a production site. The header information we will use is as follows:

```
<?php
// no direct access
defined( '_JEXEC' ) or die( 'Restricted access' );
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
 TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php echo $this->language;
 ?>" lang="<?php echo $this->language; ?>" >

<head>

<jdoc:include type="head" />
```

```
<link rel="stylesheet" href="templates/system/css/system.css" type="text/css" />
<link rel="stylesheet" href="templates/system/css/general.css" type="text/css" />
<link rel="stylesheet" href="templates/<?php echo $this->template ?>/css/template.
css" type="text/css" />

</head>
```

What does all that mean?

We have already discussed the implications of the DOCTYPE statement in the `index.php` file. The `<?php echo $this->language; ?>` is pulling the language from the site Global Configuration.

The next line is to include more header information:

```
<jdoc:include type="head" />
```

This is all header information that is set in the Global Configuration again. It includes the following tags (in a default installation):

```
<title>Welcome to the Frontpage</title>
  <meta name="description" content="Joomla! - the dynamic portal engine and
  content management system" />
  <meta name="generator" content="Joomla! 1.5 - Open Source Content Management" />
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta name="robots" content="index, follow" />
  <meta name="keywords" content="joomla, Joomla" />

  <link href="index.php?option=com_content&amp;view=frontpage&amp;format=feed&amp;
Itemid=1&amp;type=rss" rel="alternate" type="application/rss+xml" title="RSS 2.0"
/>
  <link href="index.php?option=com_content&amp;view=frontpage&amp;format=feed&amp
;Itemid=1&amp;type=atom" rel="alternate" type="application/atom+xml" title="Atom
1.0" />
  <script type="text/javascript" src="http://localhost/Joomla-1.5RC2/media/system/
js/mootools.js"></script>
  <script type="text/javascript" src="http://localhost/Joomla-1.5RC2/media/system/
js/caption.js"></script>
```

Much of this header information is created on the fly specific to the page (article) that someone is on. It includes a number of metatags—the favicon, RSS feed URLs, and some standard JavaScipt files.

The last lines in the header provide links to CSS files for the template:

```
<link rel="stylesheet" href="templates/system/css/system.css" type="text/css" />
<link rel="stylesheet" href="templates/system/css/general.css" type="text/css" />
<link rel="stylesheet" href="templates/<?php echo $this->template ?>/css/template.
css" type="text/css" />
```

The first two files, system.css and general.css contain some generic Joomla styles. The last one is all the CSS for the template, here called template.css. The PHP code `<?php echo $this->template ?>` will return the name of the current template. Writing it in this way rather than the actual real path makes the code more generic. When you create a new template you can just copy it (along with the whole header code) and not worry about editing anything.

The template CSS files can have any number of files, for example conditional ones for different browsers. This one targets IE6:

```
<!--[if lte IE 6]>
<link href="templates/<?php echo $this->template ?>/css/ieonly.css"
rel="stylesheet" type="text/css" />
<![endif]-->
```

This example is part of a technique to use a template parameter:

```
<link rel="stylesheet" href="templates/<?php echo $this->template ?>/css/<?php
echo $this->params->get('colorVariation'); ?>.css" type="text/css" />
```

Blank Joomla Template Body

Creating our first template will be very, very easy! Ready?

All we need to do is use Joomla statements that insert the contents of any modules and the mainbody.

```
<body>
<?php echo $mainframe->getCfg('sitename');?><br />
<jdoc:include type="module" name="breadcrumbs" />
<jdoc:include type="modules" name="top" />
<jdoc:include type="modules" name="left" />
<jdoc:include type="component" />
<jdoc:include type="modules" name="right" />
</body>
```

At this point (if you preview it), our site does not look very awe inspiring. The output is shown in Figure 9.3.

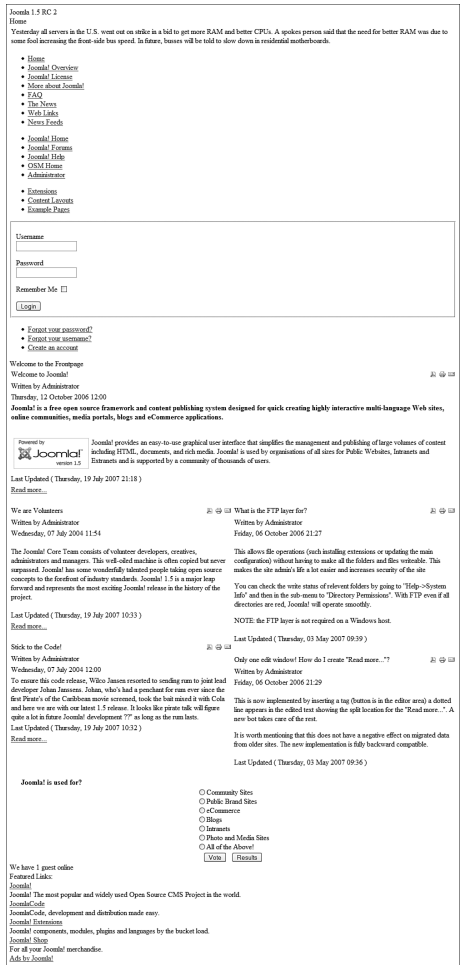

FIGURE 9.3 An unstyled template

The template contains the following in reasonably logical order:

- name of the site
- top module
- left modules
- main content
- right modules

The Least You Need to Know
The most basic template simply loads the Joomla modules and mainbody (component). Layout and design is part of the CSS, not Joomla.

The goal is to try and come as close to semantic markup as possible. From a Web point of view, it means a page can be read by anyone—a browser, a spider, or a screen reader. Semantic layout is the cornerstone of accessibility.

NOTE
What we have here really is only the potential for semantic layout. If we were to go ahead and put random modules in random locations, we would have a mess. An important consideration for CMS sites is that a template is only as good as the population of the content. It is this that often trips designers up when trying to validate their sites.

You will notice that we have used the first of a number of commands specific to Joomla to create this output:

```
<?php echo $mainframe->getCfg('sitename');?><br />
<jdoc:include type="module" name="breadcrumbs" />
<jdoc:include type="modules" name="top" />
<jdoc:include type="modules" name="left" />
<jdoc:include type="component" />
<jdoc:include type="modules" name="right" />
```

The PHP echo statement simply outputs a string from the `configuration.php` file. Here, we are using the site name; we could have as easily had the following:

```
The name of this site is <?php echo $mainframe->getCfg('sitename');?><br />
The administrator email is <?php echo $mainframe->getCfg('mailfrom');?><br />
This template is in the <?php echo $this->template?> directory<br />
The URL is <?php echo JURI::base();;?>
```

The `jdoc` statement inserts various types of (X)HTML output from modules of components.

This line inserts the output from a component. What component it will be is determined by the menu link:

```
<jdoc:include type="component" />
```

NOTE
Interestingly enough, you seem to be able to have multiple instances of component output. Not sure why you would want to, but I thought I would let you know! Might be a bug.

This line inserts the output for a module location:

```
<jdoc:include type="modules" name="right" />
```

The full syntax is actually

```
<jdoc:include type="modules" name="LOCATION" style="OPTION" />
```

We look at the various options for styles in the section about modules later in this chapter.

CSSTemplateTutorialStep1

At this point, we have a very bare template. I have created an installable template that is available from www.joomlabook.com: CSSTemplateTutorialStep1.zip.

This will install a template that has only two files, the `index.php` and `templateDetails.xml`. I removed references to other files to give a bare bones output with no CSS. This is actually a useful diagnostic template; you can install it and track errors that are occurring with a component or module.

Using CSS to Create a Tableless Layout

We will be using pure CSS to make a 3-column layout for the Joomla template. We will also be making it a fluid layout. There are two main types of web page layout—fixed and fluid—and they both refer to how the width of the page is controlled.

The width of the page is an issue because of the many browser resolutions at which people surf the Web. Although the percentage is dropping, about 17% of surfers are using an 800x600 resolution. The majority, 79%, are using 1024x768 and higher[2]. Making a fluid layout means that your valuable web page won't be a narrow column in the 1024 resolution and will be visible in full on smaller monitors.

A typical design might use tables to lay out the page. They are useful in that you just have to set the width of the columns as percentages, but they have several drawbacks.

For example, tables have lots of extra code compared to CSS layouts. This leads to longer load times (which surfers don't like) and poorer performance in search engines. The code can roughly double in size, not just with markup but also with something called "spacer gifs."

Even big companies sometimes fall into the table trap, as seen by a recent controversy about the new disney.co.uk website[3]:

There are a couple of major problems with a site that uses tables for layout.

- They are difficult to maintain. To change something you have to figure out what all the table tags like td/tr are doing. With CSS there are just a few lines to inspect.
- The content cannot be source-ordered. Many Web surfers do not see web pages on a browser. Those viewing with a text browser or screen reader will read the page from the top left corner to the bottom right. This means that they first view everything in the header and left column (for a 3-column layout) before they get to the middle column, the important stuff. A CSS layout, on the other hand, allows for "source-ordered" content, which means the content can be rearranged in the code/source. Perhaps your most important site visitor is Google, and it uses a screen reader for all intents and purposes.

Let's look at our layout using CSS. You can position elements (stuff) in several ways using CSS. For a quick introduction, a good source is Brainjar's CSS Positioning.[4]

If you are new to CSS, you might read at least one beginner's guide to CSS. Here are a few suggestions:

- Kevin Hale's *An Overview of Current CSS Layout Techniques*
  http://particletree.com/features/an-overview-of-current-css-layout-techniques/
- htmldog's *CSS Beginner's Guide*
  www.htmldog.com/guides/cssbeginner/
- yourhtmlsource.com
  www.yourhtmlsource.com/stylesheets/

The Least You Need to Know

Modern web design uses CSS rather than tables to position elements. It's difficult to learn, but worth the investment. There are many (non-Joomla) resources available to help you.

We will be using float to position our content. At its most basic, the template might look like Figure 9.4.

Still not very exciting, but let's see what the different parts are all about.

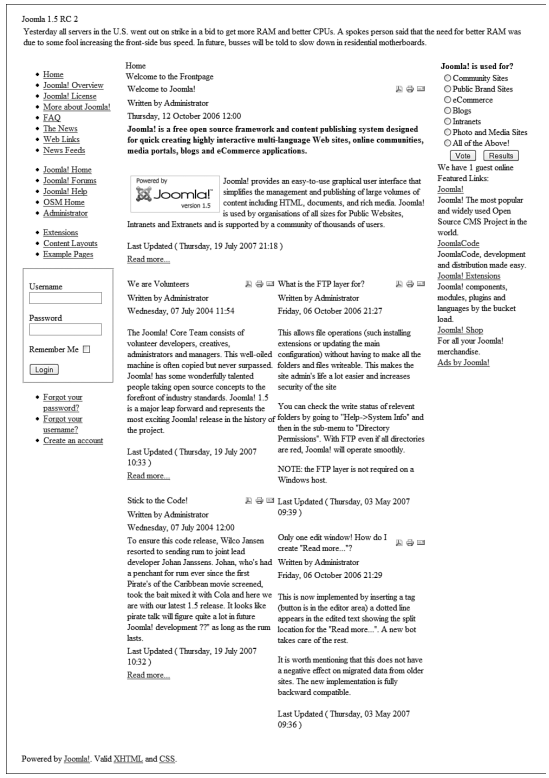

FIGURE 9.4 Basic template layout

In Figure 9.4, the left, middle, and right columns are each given their own element. Each is floated left and given a percent width that together add up to 100%. The `clear:both` style on the footer tells the browser to "stop floating" and makes the footer stretch across all three columns. When we build our second template in this chapter, we will have to use a more advanced clearing technique.

To improve the layout and to add some breathing room to the content, we need to add some column spacing, commonly called "gutter." Unfortunately, there is a problem here. You might know that Internet Explorer does not interpret CSS correctly. One problem is that it calculates width differently. We can solve this problem by not using any padding or borders on something that has a width. To get our gutter, we add another `<div>` element inside the columns.

To the CSS we add

```
.inside {padding:10px;}
```

Our resulting `<body>` code for `index.php` is:

```
<body>
<div id="wrap">
  <div id="header">
    <div class="inside">
        <?php echo $mainframe->getCfg('sitename');?>
      <jdoc:include type="modules" name="top" />
    </div>
  </div>
  <div id="sidebar">
    <div class="inside">
      <jdoc:include type="modules" name="left" />
    </div>
  </div>
  <div id="content">
    <div class="inside">
      <jdoc:include type="component" />
    </div>
  </div>
  <div id="sidebar-2">
    <div class="inside">
      <jdoc:include type="modules" name="right" />
    </div>
  </div>
  <div id="footer">
    <div class="inside">
Powered by <a href="http://joomla.org">Joomla!</a>. Valid <a href="http://valida-
tor.w3.org/check/referer">XHTML</a> and <a href="http://jigsaw.w3.org/css-valida-
tor/check/referer">CSS</a>. </div>
  </div>
</div>
<!--end of wrap-->
</body>
```

Our `template.css` file looks like this:

```
/*Compass Design layout.css CSS file*/
body {
}
#wrap {
```

```
min-width:760px;
max-width:960px;
}
#header {}
#sidebar {float:left;width:20%; overflow:hidden }
#content {float:left;width:60%; overflow:hidden }
#sidebar-2 {float:left;width:20%; overflow:hidden }
#footer {clear:both;}
.inside {padding:10px;}
```

TIP
CSS Shorthand
It's possible to reduce the amount of CSS code by using "shorthand." One example of this is padding and margin styles applied to an element, where

```
margin-top:5px; margin-bottom:5px; margin-left:10px; margin-right:10px;
```

can be replaced by:

```
margin: 5px 10px;
```

There are "shorthand" styles at the beginning of each style definition. After you have figured out the styles, fill the shorthand versions in and delete the long versions. The syntax is

```
font: font-size |font-style | font-variant | font-weight | line-height |
font-family
```

Here is an example. Rather than using this

```
font-size:1em; font-family:Arial,Helvetica,sans-serif; font-style:italic;
font-weight:bold; line-height:1.3em;
```

use this

```
font:bold 1em/1.3em Arial,Helvetica,sans-serif italic;
```

Read more about this syntax at An Introduction to CSS shorthand properties (http://home.no.net/junjun/html/shorthand.html).

This simple layout is a good one to use for learning about how to use CSS with Joomla because it shows two of the advantages of CSS over table-based layouts, it is less code, and it is easier to maintain. However, it is not source-ordered. For that we must use a more advanced layout known as a *nested float*.

Source-ordered layouts perform better for SEO than ones where the important content occurs late in the code. From a Joomla site perspective, the important content is that which is coming from the component.

Default CSS

So far, all of our CSS has been only about layout, which will make a plain page. So let's add some formatting:

```
/* layout.css CSS file*/
body {
text-align:center; /*center hack*/
}
#wrap {
min-width:760px;
max-width:960px;
width: auto !important; /*IE6 hack*/
width:960px; /*IE6 hack*/
margin:0 auto; /*center hack*/
text-align:left; /*center hack*/
}
#header {}
#sidebar {float:left;width:20%; overflow:hidden }
#content {float:left;width:60%; overflow:hidden }
#sidebar-2 {float:left;width:20%; overflow:hidden }
#footer {clear:both;}
.inside {padding:10px;}
```

We have centered the page by using a small hack. This has to be done because Internet Explorer does not read CSS accurately. With a standards-compliant browser, we could just say `margin:0` 10%; to center the page, but IE does not recognize that, so we center the "text" of the whole page and then align it left in the columns.

In celebration of IE7's support of min/max width (which IE6 does not), we can add in a minimum and maximum width. Note we have to add a tiny hack for IE6 as it does not understand these. It will ignore the `!important` statement and have a plain, old 960px width.

NOTE

It might seem strange to define our columns in percentage widths and then have a containing `div` that is fixed. Well, a few things are going on here:

- Having fluid columns inside a fixed width container makes the template very flexible. If we add width changer buttons, we only need to change one value.
- We still have a max-width so why not go all fluid? Many viewers on the Web now have enormous screens. Usability research tells us that lines of text over 900px wide are hard to read because the eyes have to go a long way to go to the next line. Limiting the fluidity makes the site more useable/accessible.

We have also added a new style to the columns: `overflow:hidden`. This will make the page "break" more consistent as we reduce its width.

At the beginning of the typography, with CSS we will set some overall styles and have what is known as a *global reset*:

```
/*Compass Design typography css */
* {
margin:0;
padding:0;
}
h1,h2,h3,h4,h5,h6,p,blockquote,form,label,ul,ol,dl,fieldset,address {
margin: 0.5em 0;
}
li,dd {
margin-left:1em;
}
fieldset {
padding:.5em;
}
body {
font-size:76%;
font-family:Verdana, Arial, Helvetica, sans-serif;
line-height:1.3;
}
```

Everything is given a zero margin and padding, and then all block level elements are given a bottom margin. This helps achieve browser consistency. You can read more about the global reset at clagnut[5] and left-justified.[6]

The font size is set to 76%. The reason for this is to try and get more consistent font sizes across browsers. All font sizes are then set in em. Having `line-height:1.3` helps readability. This means that the pages will be more accessible because the viewer will be able to resize the fonts to their own preferences. This is discussed more at "An experiment in typography" at The Noodle Incident (Owen Briggs).[7]

If we add some background colors to the header, sidebars, and content containers, we see something like what is shown in Figure 9.5.

Notice that the side columns do not reach their footer. This is because they only extend as far as their content; where the space is white on the left and on the right, they don't exist.

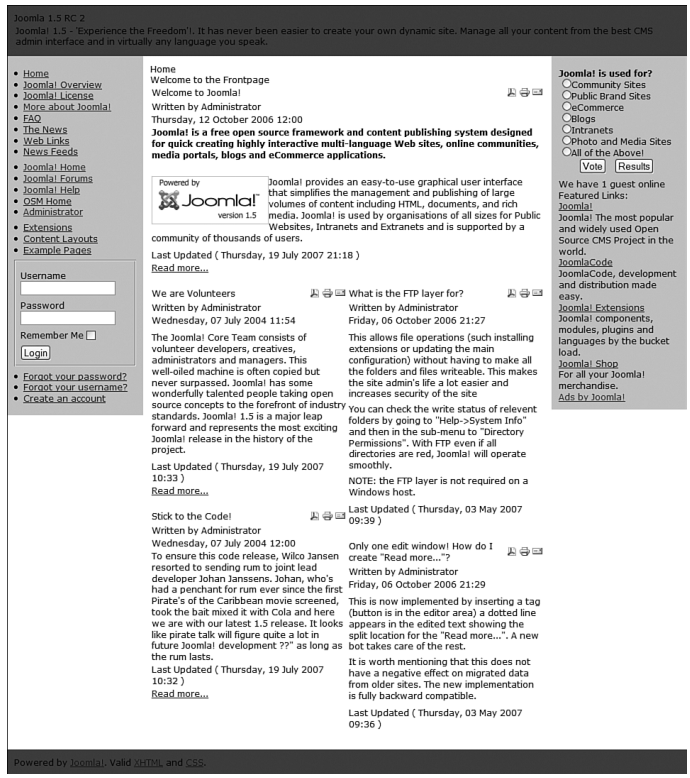

FIGURE 9.5 Basic template with typography

If we have a template that has a white background for all three columns, this is no problem. We will use this approach and will have boxes around the modules. If we want equal height columns that are colored or have boxes, we have to use a background image that will tile vertically. This technique is called *Faux Columns* and is described by Douglas Bowman[8] and Eric Meyer.[9]

Joomla-Specific CSS

Although Joomla 1.5 has the functionality to override the core output in a template, its default rendering still uses significant tables to output content in the main body. Along with these tables, CSS output is available for a designer to style pages. Based on some research by various community members, Table 9.2 shows the current list. Note, it does not include generic web page styles like H1, H2, p, ul, a, form, and so on.

TABLE 9.2 Legacy Default CSS Styles from 1.0 in 1.5

article_separator	contentpane	outline
adminform	contentpaneopen	pagenav
article_separator	contenttoc	pagenav_next
author	createdate	pagenav_prev
bannerfooter	created-date	pagenavbar
bannergroup	date	pagenavcounter
bannerheader	input	pathway
banneritem	inputbox	pollstableborder
blog	intro	read
blog_more	latestnews	search
blogsection	loclink	searchintro
breadcrumbs	mainlevel	sections
button	message	sectiontable_footer
buttonheading	metadata	sectiontableentry
clr	modifydate	sectiontablefooter
componentheading	module	sectiontableheader
content_email	moduletable	small
content_rating	mosimage	smalldark
content_vote	mosimage_caption	sublevel
contentdescription	mostread	title
contentheading	newsfeed	wrapper
contentpagetitle		

Many designs you might see in Table 9.2 actually have given CSS styles that are more specific in their definitions. Basically, a more specific rule overrides a less specific rule.

For example

```
a {color:blue;}
a:link {color:red;}

.contentheading {color:blue;}
div.contentheading {color:red;}
```

The color on a link and the color of the `.contentheading` will be *red*, as that rule is more specific (as `.contentheading` is contained within a `<div>`)

In the case of Joomla templates, you will often see more specific rules used. This often occurs when the class is on a table. Here are more examples:

```
.moduletable
table.moduletable
```

`.moduletable` is the name of the `<div>` that wraps a module. `table.moduletable` will only apply the style to a table with `class="moduletable"` on it.

`.moduletable` will apply the style regardless of what element the class is on.

```
a.contentpagetitle:link
.contentpagetitle a:link
```

`a.contentpagetitle:link` will apply the style to any `a` tags with a `.contentpagetitle` class on them that is a link.

`.contentpagetitle a:link` will apply the style to any elements *inside* `.contentpagetitle` that are links.

Specificity is not easy to understand; its often easier to start by using the most general style possible and then getting more specific if the results are not what you expect.

Here are some links to websites that discuss specificity in detail:

- www.htmldog.com/guides/cssadvanced/specificity/
- www.meyerweb.com/eric/css/link-specificity.html
- www.stuffandnonsense.co.uk/archives/css_specificity_wars.html

At the moment, our template is using several tables. As mentioned earlier, this slows the pages down and makes them harder to update. To reduce the number of tables, when we call the modules, we need to use style parameters in the `jdoc:include`.

The Least You Need to Know
Joomla will output specific elements, ids, and classes in the code of a webpage. These can be predicted and used to style the design using CSS.

Modules in Templates

When a module is called in the `index.php`, it has several options on how it is displayed.

The syntax is

```
<jdoc:include type="modules" name="LOCATION" style="OPTION" />
```

The style is optional and is defined in `templates/system/html/modules.php`. Currently, the default `modules.php` file contains the following layouts.

`OPTION="table"` (default display) modules are displayed in a column. The following shows an example of the output:

```
 <table cellpadding="0" cellspacing="0" class="moduletable<?php echo $params-
>get('moduleclass_sfx'); ?>">
 <?php if ($module->showtitle != 0) : ?>
  <tr>
   <th valign="top">
    <?php echo $module->title; ?>
   </th>
  </tr>
 <?php endif; ?>
  <tr>
   <td>
    <?php echo $module->content; ?>
   </td>
  </tr>
  </table>
```

`OPTION="horz"` makes the modules appear horizontal. Each module is output in the cell of a wrapper table. The following shows an example of the output:

```
 <table cellspacing="1" cellpadding="0" border="0" width="100%">
  <tr>
   <td valign="top">
    <?php modChrome_table($module, $params, $attribs); ?>
   </td>
  </tr>
 </table>
```

`OPTION="xhtml"` makes modules appear as a simple `div` element. The following shows an example of the output:

```
  <div class="moduletable<?php echo $params->get('moduleclass_sfx'); ?>">
  <?php if ($module->showtitle != 0) : ?>
   <h3><?php echo $module->title; ?></h3>
  <?php endif; ?>
   <?php echo $module->content; ?>
  </div>
```

`OPTION="rounded"` makes modules appear in a format that allows, for example, stretchable rounded corners. If this `$style` is used, the name of the `<div>` changes from `moduletable` to `module`. The following shows an example of the output:

```
  <div class="module<?php echo $params->get('moduleclass_sfx'); ?>">
   <div>
    <div>
```

```
    <div>
     <?php if ($module->showtitle != 0) : ?>
      <h3><?php echo $module->title; ?></h3>
     <?php endif; ?>
    <?php echo $module->content; ?>
    </div>
   </div>
  </div>
 </div>
```

`OPTION="none"` makes modules appear as raw output containing no element and no title. Here is an example:

```
echo $module->content;
```

As you can see, the CSS options ((X)HTML and rounded) are much leaner in code, which makes it easier to style the web pages. I don't recommend using suffixes of `table` (default) or `horz` unless absolutely needed.

Here's the really good bit!

If you examine the `modules.php` file, you will see all the options that exist for modules. It's easy to add your own; this is part of the new templating power that is in 1.5.We look at this in more detail in our section on template overrides.

To develop our template, we will put a module style of `"xhtml"` on all of our modules:

```
<body>
<div id="wrap">
  <div id="header">
    <div class="inside">
        <h1><?php echo $mainframe->getCfg('sitename');?></h1>
      <jdoc:include type="modules" name="top" style="xhtml" />
    </div>
  </div>
  <div id="sidebar">
    <div class="inside">
      <jdoc:include type="modules" name="left" style="xhtml" />
    </div>
  </div>
  <div id="content">
    <div class="inside">
      <jdoc:include type="module" name="breadcrumbs" style="none" />
      <jdoc:include type="component" />
    </div>
```

```
  </div>
  <div id="sidebar-2">
    <div class="inside">
      <jdoc:include type="modules" name="right" style="xhtml" />
    </div>
  </div>
  <div id="footer">
    <div class="inside">
      <jdoc:include type="modules" name="footer" style="xhtml" />
    </div>
  </div>
<!--end of wrap-->
</body>
```

Note that we cannot put these module styles on the `<jdoc:include type="component" />` because it is not a module.

The Least You Need to Know
In 1.5, the output of modules can be completely customized, or you can use the pre-built output. All these options are called module *chrome*.

We have also placed the site title inside an `<H1>` tag. It's more semantically correct and will also help in SEO. Let's also remove the background from the layout `divs`.

We add some CSS to style the modules with a border and a background for the module titles.

Our CSS now looks like this:

```
/*Compass Design typography CSS*/

* {
margin:0;
padding:0;
}
h1,h2,h3,h4,h5,h6,p,blockquote,form,label,ul,ol,dl,fieldset,address {
margin: 0.5em 0;
}
li,dd {
margin-left:1em;
}
fieldset {
padding:.5em;
}
body {
```

```
font-size:76%;
font-family:Verdana, Arial, Helvetica, sans-serif;
line-height:1.3;
margin:1em 0;
}
#wrap{
border:1px solid #999;
}
#header{
border-bottom: 1px solid #999;
}
#footer{
border-top: 1px solid #999;
}
a{
text-decoration:none;
}
a:hover{
text-decoration:underline;
}
h1,.componentheading{
font-size:1.7em;
}
h2,.contentheading{
font-size:1.5em;
}
h3{
font-size:1.3em;
}
h4{
font-size:1.2em;
}
h5{
font-size:1.1em;
}
h6{
font-size:1em;
font-weight:bold;
}
#footer,.small,.createdate,.modifydate,.mosimage_caption{
font:0.8em Arial,Helvetica,sans-serif;
color:#999;
}
.moduletable{
margin-bottom:1em;
```

```
padding:0 10px; /*padding for inside text*/ border:1px #CCC solid;
}
.moduletable h3{
background:#666;
color:#fff;
padding:0.25em 0;
text-align:center;
font-size:1.1em;
margin:0 -10px 0.5em -10px;
/*negative padding to pull h3 back out from .moduletable padding*/ }
```

NOTE
Several of the menus in the default installation have a menu suffix in the module properties of `_menu`. To get everything behaving properly, I deleted that parameter.

This typography CSS now produces the result shown in Figure 9.6.

FIGURE 9.6 Basic template with module title styling

Menus in Templates

We saw in Chapter 5, "Creating Menus and Navigation," that there are a number of settings for how a menu will be rendered.

Again, using CSS lists rather than tables results in reduced code and easier markup. After setting all our menus to lists we have only 12 tables (we see how to remove the rest using the new version 1.5 override feature). Remember, the *list* setting is the new 1.5 version; *flat list* is from 1.0 and will be depreciated. Lists are also better than tables because text-based browsers, screen readers, non-CSS supporting browsers, browsers with CSS turned off, and search bots will be able to access your content more easily.

One of the other advantages of using CSS for menus is that there is a lot of example code on various CSS developer sites. Let's look at one of them and see how it can be used.

A web page at maxdesign.com[10] has a selection of over 30 menus, all using the same underlying code. It's called the Listamatic. There is a slight difference in the code that we need to change in order to adapt these menus to Joomla.

These lists use the following code:

```
<div id="navcontainer">
<ul id="navlist">
<li id="active"><a href=" #" id="current">Item one</a></li>
<li><a href="#">Item two</a></li>
<li><a xhref="#">Item three</a></li>
<li><a href="#">Item four</a></li>
<li><a href="#">Item five</a></li>
</ul>
</div>
```

This means that there is an enclosing `<div>` called navcontainer, and the `<ul>` has an id of `navlist`. To duplicate this effect in Joomla, we need to have some sort of enclosing `<div>`.

We can achieve this by using module suffixes. If you recall, the output of an (X)HTML style option module is

```
<div class="moduletable">
  <h3>modChrome_xhtml</h3>
  modChrome_xhtml </div>
```

If we add a module suffix, that will get added to the `moduletable` class, like this:

```
<div class="moduletablesuffix">
  <h3>modChrome_xhtml</h3>
  modChrome_xhtml </div>
```

So when picking a menu from Listamatic, you need to replace the `navcontainer` class style in the CSS by `moduletablesuffix`.

NOTE
Module suffixes, to a certain extent, blur the line between site design and site administration. One of the goals of further development of the Joomla core is to clearly separate these roles. The implication is that it is likely they might be depreciated in future versions beyond 1.5.

This use of a module class suffix is helpful. It allows different colored boxes with just a simple change of the module class suffix.

The Least You Need to Know
It's best to always use the bulleted or flat list for menu output. You can then make use of many free resources for the CSS that are available on the Web.

For our site, we use List 10 by Mark Newhouse.[11] Our CSS is

```
.moduletablemenu{
padding:0;
color: #333;
margin-bottom:1em;
}
.moduletablemenu h3 {
background:#666;
color:#fff;
padding:0.25em 0;
text-align:center;
font-size:1.1em;
margin:0;
border-bottom:1px solid #fff;
}
.moduletablemenu ul{
list-style: none;
margin: 0;
padding: 0;
}
```

```
.moduletablemenu li{
border-bottom: 1px solid #ccc;
margin: 0;
}
.moduletablemenu li a{
display: block;
padding: 3px 5px 3px 0.5em;
border-left: 10px solid #333;
border-right: 10px solid #9D9D9D;
background-color:#666;
color: #fff;
text-decoration: none;
}
html>body .moduletablemenu li a {
width: auto;
}
.moduletablemenu li a:hover,a#active_menu:link,a#active_menu:visited{
border-left: 10px solid #1c64d1;
border-right: 10px solid #5ba3e0;
background-color: #2586d7;
color: #fff;
}
```

We then need to add the module suffix of *menu* (no underscore in this case) to any modules of menus we want to be styled. This produces a menu like what's shown in Figure 9.7.

For any menu we want to be styled this way, we have to add "menu" as a module suffix.

TIP

When trying to get a particular menu to work, here is a useful tip: Create a default Joomla installation and then look at the code that makes up the mainmenu. Copy and paste this code into an HTML editor (like Dreamweaver). Replace all the links by "#," and then you can add CSS rules until the effect you want is achieved. The code for the menu to create the style is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Untitled Document</title>
<style type="text/css">
<!--
.astyle {
```

continues

```
}
-->
</style>
</head>
<body>
<div class="moduletable">
<h3>Main Menu</h3>
<ul class="mainmenu">
  <li id="current" class="item1 active"><a href="#">Home</a></li>
  <li class="item2"><a href="#">Joomla! Overview</a></li>
  <li class="item3"><a href="#">What's New in 1.5?</a></li>
  <li class="item4"><a href="#">Joomla! License</a></li>
  <li class="item5"><a href="#">More about Joomla!</a></li>
  <li class="item6"><a href="#">FAQ</a></li>
  <li class="item7"><a href="#">The News</a></li>
  <li class="item8"><a href="#">Web Links</a></li>
  <li class="item9"><a href="#">News Feeds</a></li>
</ul>
</div>
</body>
</html>
```

The CSS is embedded instead of linked to make editing easier.

FIGURE 9.7 Basic template with menu styling

Hiding Columns

So far, we have our layout such that we always have three columns, regardless of whether there is any content included. From the perspective of a CMS template, this is not very useful. In a static site the content would never change, but we want to give our site administrators the ability to put their content anywhere they want to without having to worry about editing CSS layouts. We want to be able to "turn off" a column automatically or "collapse" it if there is no content there.

During the development of the Joomla 1.5 templating engine, there were a number of changes and improvements. Quoting directly from the Joomla development blog[12]:

> The changes to the template system in Joomla 1.5 can be divided into two categories. First, there are changes to the way things where done in Joomla 1.0—for example the new way modules are loaded, and second there are also a bunch of extra features, like template parameters…a quick overview:
>
> Changes to the old ways
>
> **mosCountMoules**
>
> The mosCountModules function has been replaced by the $this->countModules function and support for conditions has been added. This allows designers to easily count the total number of modules in multiple template positions in just one line of code, for example $this->countModules('user1 + user2); which will return the total number of modules in position user1 and user2.

NOTE
More information is also available in the Joomla forum.[13]

So the general use of `mosCountModules` would be

```
<?php if($this->countModules('condition')) : ?>
   do something
<?php else : ?>
   do something else
<?php endif; ?>
```

There are four possible conditions. As an example, let's count the number of modules in Figure 9.7. We could insert this code somewhere in the `index.php`:

```
left=<?php echo $this->countModules('left');?><br />
left and right=<?php echo $this->countModules('left and right');?><br />
```

```
left or right=<?php echo $this->countModules('left or right');?><br />
left + right=<?php echo $this->countModules('left + right');?>
```

- **`countModules('left')`**. Will return 4; there are 4 modules on the left.
- **`countModules('left and right')`**. Will return 1; there is a module in the left and right-hand position.
- **`countModules('left or right')`**. Will return 1; there is a module in the left or the right-hand position.
- **`countModules('left + right')`**. Will return 7; counting the modules in the left and right-hand positions.

In this situation, we need to use the function that allows us to count the modules present in a specific location. So for example, if there is no content published in the right column, we can adjust the column sizes to fill that space.

There are several ways to do this. We could put the conditional statement in the body to not show the content and then have a different style for the content based on what columns were there. To make it as easy as possible, I have a series of conditional statements in the head tag that (re)define some CSS styles:

```
<?php
if($this->countModules('left and right') == 0) $contentwidth = "100";
if($this->countModules('left or right') == 1) $contentwidth = "80";
if($this->countModules('left and right') == 1) $contentwidth = "60";
?>
```

So we count:

- If there is nothing in left OR right, we are 100%.
- If there is something in left OR right, we are 80%.
- If there is something in left AND something in right, we are 60%.

We then need to change the `index.php` file in the content `div` to

```
<div id="content<?php echo $contentwidth; ?>">
```

Change the `layout css` to

```
#content60 {float:left;width:60%;overflow:hidden;}
#content80 {float:left;width:80%;overflow:hidden;}
#content100 {float:left;width:100%;overflow:hidden;}
```

The PHP conditional statements in the head must appear *after* the line that links to the `template.css` file. This is because if there are two identical CSS style rules; the one that is last will overwrite all the others.

This can also be done in a similar fashion by having the `if` statement import a sub CSS file.

TIP

While you try to troubleshoot your conditional statements, you can add a line of code into your `index.php`, like this, to show what the value is:

```
This content column is <?php echo $contentwidth; ?>% wide
```

So we are half-way there, but now we have empty `div` containers where the columns are.

Hiding Module Code

When creating collapsible columns, it is good practice to set up the modules not to be generated if there is no content there. If this is not done, the pages will have empty `<div>`s in them, which can lead to cross browser issues.

To hide the empty `<div>`, the following `if` statement is used:

```
<?php if($this->countModules('left')) : ?>
<div id="sidebar">
  <div class="inside">
    <jdoc:include type="modules" name="left" style="xhtml" />
  </div>
</div>
<?php endif; ?>
```

Using this code, if there is nothing published in the left, then `<div id="sidebar">` will not be outputted.

Using these techniques for our left and right columns, our `index.php` file now looks like the following code. We will also add an "include for the breadcrumbs module," the module that shows the current page and pathway. Note that this now needs to be included in the `index.php` file and also published as a module.

```
<?php
// no direct access
defined( '_JEXEC' ) or die( 'Restricted access' );
```

```
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php echo $this->language;
?>" lang="<?php echo $this->language; ?>" >

<head>

<jdoc:include type="head" />

<link rel="stylesheet" href="templates/system/css/system.css" type="text/css" />
<link rel="stylesheet" href="templates/system/css/general.css" type="text/css" />
<link rel="stylesheet" href="templates/<?php echo $this->template
?>/css/template.css" type="text/css" />

<?php
if($this->countModules('left and right') == 0) $contentwidth = "100";
if($this->countModules('left or right') == 1) $contentwidth = "80";
if($this->countModules('left and right') == 1) $contentwidth = "60";
?>

</head>

<body>
<div id="wrap">
  <div id="header">
    <div class="inside">
        <h1><?php echo $mainframe->getCfg('sitename');?></h1>
      <jdoc:include type="modules" name="top" style="xhtml" />
    </div>
  </div>
  <?php if($this->countModules('left')) : ?>
  <div id="sidebar">
    <div class="inside">
      <jdoc:include type="modules" name="left" style="xhtml" />
    </div>
  </div>
  <?php endif; ?>

<div id="content<?php echo $contentwidth; ?>">
    <div class="inside">
      <jdoc:include type="module" name="breadcrumbs" style="none" />
      <jdoc:include type="component" />
    </div>
```

```
  </div>
<?php if($this->countModules('right')) : ?>
  <div id="sidebar-2">
    <div class="inside">
      <jdoc:include type="modules" name="right" style="xhtml" />
    </div>
  </div>
  <?php endif; ?>
  <?php if($this->countModules('footer')) : ?>
  <div id="footer">
    <div class="inside">
      <jdoc:include type="modules" name="footer" style="xhtml" />
    </div>
  </div>
  <?php endif; ?>
<!--end of wrap-->
</body>
</html>
```

The Least You Need to Know
Elements such as columns or module locations can be hidden (or collapsed) when there is no content in them. This is done using conditional PHP statements that are linked to different CSS styles.

I would recommend a slightly different way of producing the footer. In the manner shown here, it is hard coded into the `index.php` file, which makes it hard to change. Right now, the "footer" module in the administrative backend shows the Joomla copyright and can't be easily edited. It makes much more sense to have a custom (X)HTML module placed in the footer location so the site administrator can change it more easily. If you wanted to create your own footer, you would simply unpublish that module and create a custom html module with whatever language you wanted.

In this case we would replace

```
<jdoc:include type="modules" name="footer" style="xhtml" />
```

with

```
<jdoc:include type="modules" name="bottom" style="xhtml" />
```

We must also remember to add this position to the `templateDetails.xml` file.

TIP

There are several names associated with modules in Joomla: banner, left, right, user1, footer, and so on. One important thing to realize is that the names do not correspond to any particular location. The location of a module is completely controlled by the template designer, as we have seen. It's customary to place them in a location that is connected to the name, but it is not required.

This basic template shows some of the fundamental principles of creating a Joomla template.

CSSTemplateTutorialStep2

We now have a basic, but functional template. Some simple typography has been added, but more importantly, we have created a pure CSS layout that has dynamic collapsible columns. I have created an installable template that is available from www.joomlabook.com: CSSTemplateTutorialStep2.zip.

Now that we have the basics finished, let's create a *slightly* more attractive template using the techniques we have learned.

Making a Real Joomla 1.5 Template

The first thing we need to start with is our *comp*. A comp is the design that is the basis of the template. We use one kindly donated by Casey Lee, the Lead Designer from Joomlashack[14] for our purposes. It's called "Bold," and we can see it in Figure 9.8.

Slicing and Dicing

The next step in the process is what is known as slicing. We need to use our graphics program to create small sliced images that can be used in the template. It's important to pay attention to how the elements can resize if needed. (My graphics application of choice is Fireworks, because I find it better suited to web design—as opposed to print—than Photoshop).

FIGURE 9.8 A design comp from Joomlashack

Setting Up Module Locations

This template will have some specific locations for specific modules, slightly different from the standard Joomla installation. To make sure the modules are correctly set up as you work through this template, make sure of the following:

- User1=for the search module
- User2=for the top menu
- Top=for newsflash or custom HTML module

Nothing else should be published in these locations.

Header

The header image has a faint swish at the top. We want to retain that, so we put the image in as a background and then assign a color also. That way, the header will scale vertically if we need it to; for example, if the font sizes are resized. We also need to change the colors of any type to white so they show up on the black background.

We also use the background image for the search box. We need to make sure that we target the correct input by using CSS specificity. I have also used absolute positioning inside a relatively positioned element to place the search box where I want it. The image will not scale with text resizing with just a single image. That would require a top and bottom image. That's another exercise for you!

```
#header {
color:#fff;
background:#212121 url(../images/header.png) no-repeat;
position:relative;}
#header h1 {
font-family:Arial, Helvetica, sans-serif small-caps;
font-variant:small-caps;
font-stretch:expanded;
padding-left:20px;}
#header input {
background:url(../images/search.png) no-repeat;
border:0;
height:22px;
width:168px;
padding:2px;
font:1em Arial, Helvetica, sans-serif;
}
#header .search {
position:absolute;
top:20px;
right:20px;
}
```

I did not use a graphical logo here; I used plain text. The reason is mainly because SEOs, as search engines, cannot read images. One could do some nifty image replacement, but I will leave that as an exercise for you to do on your own.

Our header now looks like what's shown in Figure 9.9.

FIGURE 9.9 Header image background

Next, we need to implement a technique used to show a background on a fluid column: sliding doors.

Column Backgrounds

Recall that when we put a color background on the columns, the color did not extend all the way to the footer. This is because the `div` element, in this case sidebar and sidebar-2, is only as tall as the content. It does not grow to fill the containing element.

We have to use a technique called *Sliding Faux Columns*, with which you essentially create two wide images that will slide over each other. We need to create two new containers to hold the backgrounds. Normally, we could apply one to the `#wrap`, but I am using an extra (and wasteful) container for illustration purposes.

For a full description, you can check out these two guides:

- http://alistapart.com/articles/fauxcolumns/
- www.communitymx.com/content/article.cfm?page=1&cid=AFC58

In our case, our maximum width is 960px, so we start with an image of that width. In the image source files, it is `slidingcolumns.png`. We then export two slices (I used the same slice and just hid/revealed the side images), one 960px wide with a 192px background on the left, and one 960px wide with a 196px background on the left.

NOTE
The left image needs to have a white background, and the right image needs a transparent background. I modified the color of the backgrounds as I exported the images from the source file.

Where does 192px come from? It's 20% of 960, as our columns are 20% wide.

We use the `background-position` property to place the images in the correct place. Here, we are using condensed CSS format so they are part of the background property:

```
#leftfauxcol {
background:url(../images/leftslidingcolumn.png) 20% 0;
}
```

```
#rightfauxcol {
background:url(../images/rightslidingcolumn.png) 80% 0;
}
```

In our `index.php`, we simply added an inner container inside the wrap:

```
<div id="wrap">
  <?php if($this->countModules('left')) : ?>
  <div id="leftfauxcol">
    <?php endif; ?>
    <?php if($this->countModules('right')) : ?>
    <div id="rightfauxcol">
      <?php endif; ?>
      <div id="header">
```

We also need to put a conditional on the closing `div`s:

```
      <?php if($this->countModules('left')) : ?>
    </div>
    <!--end of leftfauxcol-->
    <?php endif; ?>
    <?php if($this->countModules('right')) : ?>
  </div>
  <!--end of rightfauxcol-->
  <?php endif; ?>
```

We must also put a background on our footer and bottom modules/elements; otherwise, the column background would be shown:

```
#footer {
background:#212121;
color:#fff;
text-align:right;
clear:both;
}
#bottom {
background:#333;
color:#666;
padding:10px 50px;
}
```

We need to clear the floats so that the float container (the faux columns) will extend to the bottom of the page. The traditional method to do this was to use the property `:after`.[15] But with the release of IE7, this method will not work completely. We need

to address clearing the floats in Internet Explorer 6 *and* 7, and this is where it all goes down the tubes.

A couple[16] of solutions have been found[17]; we are going to use the *Float (nearly) Everything* option[18] here.

Thus, we add a simple `clear:both` to the `#footer`, and we add floats to the `fauxcol` wrappers. We add these to a *conditional stylesheet* specifically for IE6:

```
#leftfauxcol {
float:left;
width:100%;
}
#rightfauxcol {
float:left;
width:100%;
}
#footer {
float:left;
width:100%;
}
```

We will have to add some conditional statements to the head of the `index.php` file:

```
<!--[if lte IE 6]>
<link href="templates/<?php echo $this->template ?>/css/ie6only.css"
rel="stylesheet" type="text/css" />
<![endif]-->
<!--[if lte IE 7]>
<link href="templates/<?php echo $this->template ?>/css/ie7only.css"
rel="stylesheet" type="text/css" />
<![endif]-->
```

Flexible Modules

In our design, we have a large initial module block. We don't know how tall the text will be that is needed. To solve that problem, we put the module `jdoc:include` statement in a containing element and give it a background of the same color as the image. This is the same strategy we used for the header:

```
<?php if($this->countModules('top')) : ?>
<div id="top">
  <div class="inside">
    <jdoc:include type="modules" name="top" style="xhtml" />
  </div>
```

```
    </div>
    <?php else : ?>
    <div id="top"> </div>
    <?php endif; ?>
```

Note, we have also used a conditional comment so that if the top module location has no content, the orange teaser image will not be there. What will be there is an empty container that will contain a little of the background image and 20px worth of vertical padding. This is purely for aesthetics.

The CSS needs to use CSS specificity to override the `moduletable` styles defined earlier:

```
#top {
background:#ea6800 url(../images/teaser.png) no-repeat;
padding:10px;
}
#top .moduletable h3 {
color:#fff;
background:none;
text-align:left;
font:2.5em Arial, Helvetica, sans-serif normal;
padding:0;
margin:0;
font-stretch:expanded
}
#top .moduletable{
font:bold 1em/1.2 Tahoma,Arial, Helvetica, sans-serif;
color:#fff;
margin:0;
padding:0;
border:0;
}
```

Now we need to focus on some of the typography.

Typography

Many of the links will need to be white, so we will define them as such globally and then modify the color for the center column:

```
a:link,a:visited {
text-decoration:underline;
color:#fff;
```

```
}
a:hover {
text-decoration:none;
}
#content60 a:link,#content60 a:visited,#content80 a:link,#content80
a:visited,#content100 a:link,#content100 a:visited {
color:#000;
}
```

The design has a stylize button. We create this using a background image from the comp. It's a thin slice that is tiled horizontally:

```
.button {
border:#000 solid 1px;
background:#fff url(../images/buttonbackground.png) repeat-x;
height:25px;
margin:4px 0;
padding:0 4px;
cursor:hand;
}
```

For tables, such as FAQ, we can add an easy background by repeating the use of the image we used for the teaser. Reusing the image is thematic and also saves on image download, making the pages load faster.

```
.sectiontableheader {
background:url(../images/teaser.png);
padding:5px;
color:#fff;
font:1.2em bold Arial, Helvetica, sans-serif;
}
```

Modules need just a simple redefinition and adjustments to the padding and margins:

```
/* Module styling */
.moduletable {
margin-bottom:1em;
color:#fff;
font-size:1.1em;
}
.moduletable h3 {
font:1.3em Tahoma,Arial,Helvetica,sans-serif;
background:#000;
color:#ccc;
```

```
text-align:left;
margin:0 -10px;
padding:5px 10px;
}
```

Menus, as always, need a lot of CSS style. Here, we keep it as simple as possible. We slice a single image that includes both the bullet and the underline, not that the styling is turned "on" by applying a module suffix of `menu` to any list of the links that we want this look applied to:

```
/*Menu Styling*/
.moduletablemenu {
margin-bottom:1em;
}
.moduletablemenu h3 {
font:1.3em Tahoma,Arial,Helvetica,sans-serif;
background:#000;
color:#ccc;
text-align:left;
margin:0 -10px;
padding:5px 10px;
}
.moduletablemenu ul {
list-style:none;
margin:5px 0;
}
.moduletablemenu li {
background:url(../images/leftmenu.png) bottom left no-repeat;
height:24px;
font:14px Tahoma,Arial, Helvetica, sans-serif;
margin:10px 0;
padding:0 0 0 10px;
}
.moduletablemenu a:link,.moduletablemenu a:visited {
color:#fff;
display:block;
text-decoration:none;
padding-left:5px;
}
.moduletablemenu a:hover {
text-decoration:none;
color:#fff;
background:#ADADAD;
}
```

Last is the Tab menu at the top right. As an accessibility advocate, we want to set this up so that the tabs will scale as the font is resizing. Fortunately, a technique has been developed to do this; it's actually the same principle we use for our columns, the sliding doors[19] again!

We will also try and do some speed optimization for the template and use just a single image for the left and right side of the "doors," as well as the on and off state. This is known as using *sprites*.[20]

The CSS is not too hard; we just have to fiddle around with the vertical position of the image background for the "on" state:

```
/*Tab Menu Styling*/
.moduletabletabs {
font:bold 1em Georgia, Verdana, Geneva, Arial, Helvetica, sans-serif;
}
.moduletabletabs ul {
list-style:none;
float:right;
margin:0;
padding:0;
background:#212121;
width:100%;
}
.moduletabletabs li {
float:right;
background:url(../images/tabs.png) no-repeat 0 -4px;
margin:0;
padding:0 0 0 12px;
}
.moduletabletabs a:link,.moduletabletabs a:visited {
float:left;
display:block;
color:#000;
background:url(../images/tabs.png) no-repeat 100% -4px;
text-decoration:none;
margin:0;
padding:7px 18px 5px 9px;
}
.moduletabletabs #current {
background:url(../images/tabs.png) no-repeat 0 -84px;
}
.moduletabletabs #current a {
color:#fff;
```

```
background:url(../images/tabs.png) no-repeat 100% -84px;
}
```

We also need to add the module suffix of `tabs` to the module for the menu we are using.

If you look back at the original design, you notice that there were icons on these tabs. As we are already using two background images, one on the li and one on the link, we would need a third element on which to place the icon background. You could do this by having a span, but this is advanced CSS Jujutsu. I'll leave that as a homework assignment.

The last thing that remains is to revise the `templateDetails.xml` file. It needs to contain all the files and images used in the template so it will install properly as a zip file. There are a number of tools that will do this automatically for you if you are using 1.0.X, but at the time of writing, none are available for 1.5.

Our finished template should look like Figure 9.10.

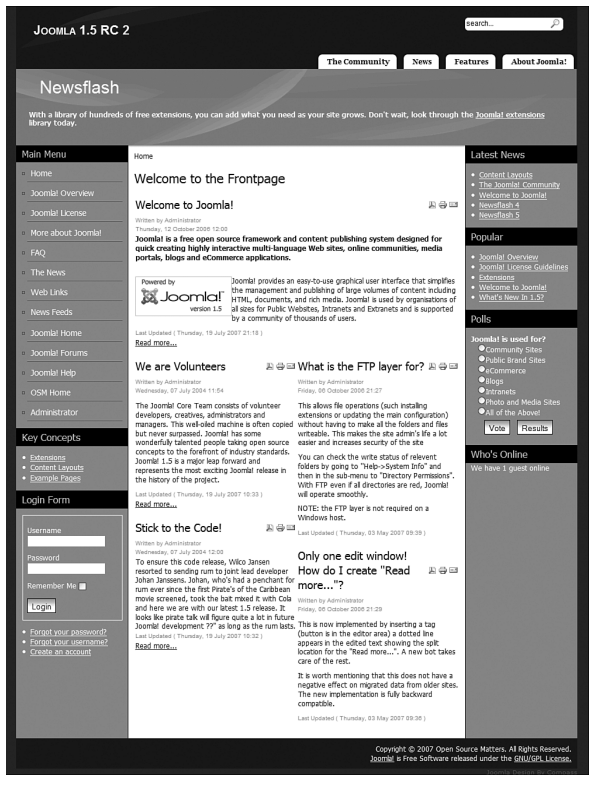

FIGURE 9.10 Advanced template with typography

The Least You Need to Know
Creating a production Joomla template is more a question of graphical design and CSS manipulation than some special "Joomla knowledge."

CSSTemplateTutorialStep3

We now have a template based on a comp (or design). Some simple typography has been added, but more importantly, we have created a pure CSS layout that has dynamic collapsible columns and a slick tabbed menu. I have created an installable template that is available from www.joomlabook.com: CSSTemplateTutorialStep3.zip.

Now that we have the basics done, let's start delving into some of the advanced features possible with 1.5 templates.

Advanced Templating Features

Joomla 1.5 offers a number of advanced template features that significantly expand what is possible with templates. We have already seen one example in this chapter, the ability to create custom *chrome* or output for modules.

Let's examine each of these in turn:

- Template Parameters
- Template Overrides

Template Parameters

New in 1.5 is the addition of template parameters for templates. This allows you to pass variables to the template from options selected in the administrative backend.

We can add a relatively simple parameter function to our template. In the `templateDetails.xml` file, add the following:

```
<params>
<param name="template_width" type="list" default="fluid" label="Template Width"
description="Width style of the template">
   <option value="fluid">Fluid with maximum and minimum</option>
   <option value="medium">Medium</option>
   <option value="small">Small</option>
  </param>
</params>
```

You also need a file called `params.ini` in your template folder. It can be a blank file, but Joomla needs this file to store what settings you have. For example, an INI file for the previous example might look like this:

```
template_width=2
```

You need to make sure that this file is writable so changes can be made.

We also need to add that as a file in the `templateDetails.xml` file.

In the Template Manager for that template, you see the settings for the parameter, as shown in Figure 9.11.

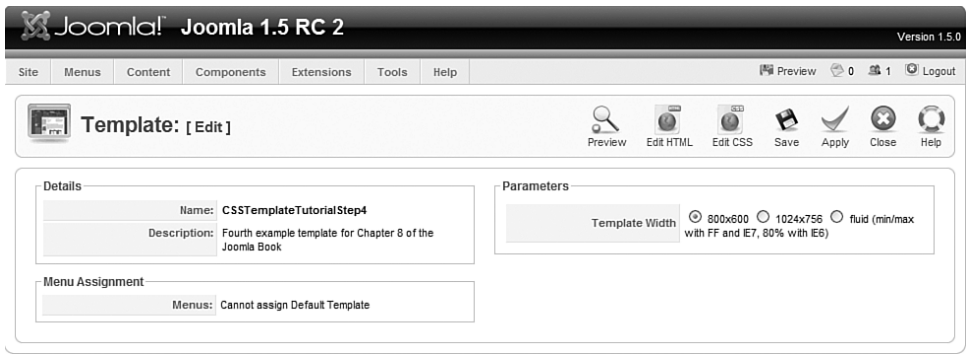

FIGURE 9.11 Template parameters in admin backend

We can see that it is a simple drop-down with three options.

```
<param name="template_width" type="radio" default="0" label="Template Width"
description="Change width setting of template">
<option value="0">800x600</option>
<option value="1">1024x756</option>
<option value="2">fluid (min/max with FF and IE7, 80% with IE6)</option>
</param>
```

Then we change the body tag in our `index.php` to the following:

```
<body class="width_<?php echo $this->params->get('template_width'); ?>">
```

We then add the following to the CSS file:

```
body.width_0 div#wrap {
width: 760px;
}
```

```
body.width_1 div#wrap {
width: 960px;
}
body.width_2 div#wrap {
min-width:760px;
max-width:960px;
width:auto !important;
width:960px;
}
#wrap {
text-align:left;
margin:0 auto;
}
```

This gives us three options: a fixed narrow width, fixed wide width, and a fluid version.

Using template parameters in this way can give the site administrator flexibility in almost any facet of a template, width, color, and so on, all controlled with conditional PHP setting CSS styles.

Template Overrides

Perhaps the most powerful new feature of templates in 1.5 is the ability to easily override core output. This is done with new output files called template files that correspond to the layout views of components and modules. Joomla checks in each case to see if one exists in the template folder, and if one does, uses that one and overrides the normal output.

Override Structure

All of the layout views and templates are in the main core in a `/tmpl/` folder. The location is slightly different for components as for modules because modules essentially have only one view. For example

```
modules/mod_newsflash/tmpl/
modules/mod_poll/tmpl/
components/com_login/views/login/tmpl/
components/com_content/views/section/tmpl/
```

The basic structure of all components and modules is View→Layout→Templates.

Table 9.3 shows some examples; note that modules only have one view.

TABLE 9.3 Example overrides

View	Layout	Templates
Category	`Blog.php`	`blog_item.php`
		`blog_links.php`
Category	`default.php`	`default_items.php`
	`default.php`	
(Newsflash module)	`horz.php`	`_item.php`
	`vert.php`	

There are usually several template files involved for a particular layout. They have a common naming convention (see Table 9.4).

TABLE 9.4 Naming convention of overrides

Filename Convention	Description	Example
`layoutname.php`	The master layout template	`blog.php`
`layoutname_templatename.php`	A child layout template called from the master layout file	`blog_item.php` `blog_links.php`
`_templatename.php`	A common layout template used by different layouts	`_item.php`

Overriding Modules

Each module has a new folder that contains its templates, which is called `tmpl`. Inside are PHP files that create the output. For example

```
/modules/mod_newsflash/tmpl/default.php
/modules/mod_newsflash/tmpl/horiz.php
/modules/mod_newsflash/tmpl/vert.php
/modules/mod_newsflash/tmpl/_item.php
```

The first three are the three layouts of Newsflash based on which module options are chosen, and the `_item.php` file is a common layout template used by all three. Opening that file, we find

```
<?php // no direct access
defined('_JEXEC') or die('Restricted access'); ?>
<?php if ($params->get('item_title')) : ?>
<table class="contentpaneopen<?php echo $params->get( 'moduleclass_sfx' ); ?>">
```

```
<tr>
      <td class="contentheading<?php echo $params->get( 'moduleclass_sfx' ); ?>"
width="100%">
      <?php if ($params->get('link_titles') && $item->linkOn != '') : ?>
            <a href="<?php echo $item->linkOn;?>" class="contentpagetitle<?php
echo $params->get( 'moduleclass_sfx' ); ?>">
                  <?php echo $item->title;?>
            </a>
       <?php else : ?>
            <?php echo $item->title; ?>
      <?php endif; ?>
      </td>
</tr>
</table>
<?php endif; ?>

<?php if (!$params->get('intro_only')) :
      echo $item->afterDisplayTitle;
endif; ?>

<?php echo $item->beforeDisplayContent; ?>

<table class="contentpaneopen<?php echo $params->get( 'moduleclass_sfx' ); ?>">
      <tr>
            <td valign="top" colspan="2"><?php echo $item->text; ?></td>
      </tr>
</table>
<?php if (isset($item->linkOn) && $item->readmore) :
      echo '<a href="'.$item->linkOn.'">'.JText::_('Read more').'</a>';
endif; ?>
```

We could change this to remove the tables to make it a little more accessible:

```
<?php // no direct access
defined('_JEXEC') or die('Restricted access'); ?>
<?php if ($params->get('item_title')) : ?>
<div class="contentpaneopen<?php echo $params->get( 'moduleclass_sfx' ); ?>">
      <div class="contentheading<?php echo $params->get( 'moduleclass_sfx' ); ?>">
      <?php if ($params->get('link_titles') && $item->linkOn != '') : ?>
            <a href="<?php echo $item->linkOn;?>"
class="contentpagetitle<?php echo $params->get( 'moduleclass_sfx' ); ?>">
                  <?php echo $item->title;?>
            </a>
       <?php else : ?>
```

```
            <?php echo $item->title; ?>
       <?php endif; ?>
       </div>
</div>
<?php endif; ?>

<?php if (!$params->get('intro_only')) :
      echo $item->afterDisplayTitle;
endif; ?>

<?php echo $item->beforeDisplayContent; ?>

<div class="contentpaneopen<?php echo $params->get( 'moduleclass_sfx' ); ?>">
<?php echo $item->text; ?>
</div>
<?php if (isset($item->linkOn) && $item->readmore) :
      echo '<a href="'.$item->linkOn.'">'.JText::_('Read more').'</a>';
endif; ?>
```

This new file should be placed in the template directory in a folder called `html` as follows:

```
templates/templatetutorial15bold/html/mod_newsflash/_item.php
```

We just took the tables out of the Newsflash module—as easy as that!

Component Overrides

Components work almost exactly the same way, except there are several views associated with many components.

If we look in the `com_content` folder, we see a folder called `views`.

```
/components/com_content/views/
/components/com_content/views/archive
/components/com_content/views/article
/components/com_content/views/category
/components/com_content/views/section
```

So these folders would match the four possible views for content, archive, article, category, and section.

Inside a view, we find the `tmpl` folder, and in that, the different layouts that are possible.

If we look in the `category` folder, we see

```
/components/com_content/views/category/blog.php
/components/com_content/views/category/blog_item.php
/components/com_content/views/category/blog_links.php
/components/com_content/views/category/default.php
/components/com_content/views/category/default_items.php
```

Note that in the case of `com_content`, the `default.php` layout is referring to the *standard* layout that presents articles as a link list.

Opening up the `blog_item.php` file we see the tables currently used. If we want to override the output, we put what we want to use in our `template/html/` folder, for example:

```
templates/templatetutorial15bold/html/com_content/category/blog_item.php
```

It's a relatively simple process to copy and paste all these views from the `/components/` and `/modules/` folders into the `templates/yourtemplate/html` folder.

The template override functionality provides a powerful mechanism to customize your Joomla site through its template. You can create output templates that focus on SEO, accessibility, or the specific needs of a client.

The Least You Need to Know
Joomla 1.5 offers new features for templates that allow designers to completely control the code and presentation of a Joomla website.

Tableless Joomla

The Joomla download also contains a template called Beez that is a developed example of the template overrides in action. The Design and Accessibility team have created a full example set of overrides as contained in the `html` folder. Our final example is a template that uses these overrides to remove all tables from the output of Joomla.

CSSTemplateTutorialStep4

We now have a template based on a comp (or design). More visual typography has been added, but more importantly, we have used our pure CSS layout to create a template that has dynamic collapsible columns and a slick tabbed menu. We have then overridden the output of Joomla so that no other tables are used. I have created an installable template that is available from www.joomlabook.com:

CSSTemplateTutorialStep4.zip

Summary

In this chapter, we worked through four examples of templates, each time building the complexity and features.

- Modern websites separate content from presentation using a technology known as Cascading Style Sheets (CSS). In Joomla, the template controls the presentation of the content.
- When creating a template, you have to have Joomla "running" on a server so you can make changes and refresh the page output.
- Creating valid templates should be a path not a goal. The idea is to make your template as accessible as possible, for humans and spiders, not to achieve a badge for valid markup.
- The most basic template simply loads the Joomla modules and mainbody (component). Layout and design are part of the CSS, not Joomla.
- Modern web design uses CSS rather than tables to position elements. It's difficult to learn but worth the investment. There are many (non-Joomla) resources available to help you.
- Joomla will output specific elements, ids, and classes in the code of a web page. These can be predicted and used to style the design using CSS.
- In 1.5, the output of modules can be completely customized, or you can use the pre-built output. All of these options are called module chrome.
- It's best to always use the bulleted or flat list for menu output. You can then make use of many free resources on the Web for the CSS.
- Elements such as columns or module locations can be hidden (or collapsed) when there is no content in them. This is done using conditional PHP statements that are linked to different CSS styles.
- Creating a production Joomla template is more a question of graphical design and CSS manipulation than some special "Joomla knowledge."
- Joomla 1.5 offers new features for templates that allow designers to completely control the code and presentation of a Joomla website.

[1] This article at Compass Design helps explain this more: www.compassdesigns.net/tutorials/joomla-tutorials/installing-joomla-doctype-and-the-blank-joomla-template.html

[2] www.upsdell.com/BrowserNews/stat_trends.htm#res

[3] www.compassdesigns.net/joomla-blog/general-joomla/what-makes-a-good-designer.html

[4] www.brainjar.com/css/positioning/

[5] www.clagnut.com/blog/1287/

[6] http://leftjustified.net/journal/2004/10/19/global-ws-reset/

[7] www.thenoodleincident.com/tutorials/typography/template.html

[8] www.stopdesign.com/log/2004/09/03/liquid-bleach.html

[9] www.meyerweb.com/eric/thoughts/2004/09/03/sliding-faux-columns/

[10] http://css.maxdesign.com.au/listamatic/index.htm

[11] http://css.maxdesign.com.au/listamatic/vertical10.htm

[12] http://dev.joomla.org/component/option,com_jd-wp/Itemid,33/p,210/

[13] http://forum.joomla.org/index.php/topic,101825.msg535479.html#msg535479

[14] www.joomlashack.com

[15] http://positioniseverything.net/easyclearing.html

[16] http://www.quirksmode.org/css/clearing.html

[17] http://www.sitepoint.com/blogs/2005/02/26/simple-clearing-of-floats/

[18] http://orderedlist.com/articles/clearing-floats-fne/

[19] www.alistapart.com/articles/slidingdoors/

[20] www.fiftyfoureleven.com/sandbox/weblog/2004/jun/doors-meet-sprites/