
# Joomla!™ 1.5: A User's Guide

## BUILDING A SUCCESSFUL JOOMLA! POWERED WEBSITE

SECOND EDITION

BARRIE M. NORTH

ISBN-13: 978-0-13-701231-2
ISBN-10: 0-13-701231-4
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing May 2009

**Editor in Chief**
Mark Taub

**Acquisitions Editor**
Debra Williams Cauley

**Development Editor**
Songlin Qiu

**Managing Editor**
Kristy Hart

**Project Editor**
Betsy Harris

**Copy Editor**
Kitty Wilson

**Indexer**
Erika Millen

**Proofreader**
Kathy Ruiz

**Technical Reviewers**
TJ Baker
David Childs
G.D. Speer

**Publishing Coordinator**
Kim Boedigheimer

**Cover Designer**
Alan Clements

**Compositor**
Nonie Ratcliff

# Preface

Joomla! is an open source content management system (CMS) that anyone can download for free (see forge.joomla.org/sf/go/projects.joomla/frs). This makes it an ideal choice for small businesses. Don't let the price tag fool you, though; Joomla! is powerful and robust, and more big organizations are choosing to use open source software solutions all the time. Its universal appeal has made Joomla! hugely popular as a CMS.

As Joomla! matures, it is being adopted by more and more organizations, from corporations to schools and universities to government organizations to newspapers and magazines to small businesses. Its greatest advantage is its flexibility. You can see it on a huge variety of sites.

## The Purpose of This Book

This book is about Joomla!, a popular and award-winning ("Best Linux/Open Source Project" for 2005) open source CMS. This book walks, step-by-step, through everything you need to develop a successful website powered by Joomla!. The book gives a general overview of management of a CMS and teaches you key concepts regarding content organization, editing, and templates. Finally, this book examines some more general topics, such as how to maximize search engine optimization (SEO) with Joomla! and what resources are available in the Joomla! web community.

This book focuses on the most current release of Joomla!, Joomla! 1.5, which is a significant update to this CMS.

## This Book's Target Audience

This book primarily targets people using Joomla! 1.5 to create a website, either for themselves or their clients. It's easy to read and low on technical jargon. It doesn't assume that you know PHP or CSS.

All the concepts in this book are explained with step-by-step contextual examples. If you follow all the steps in all the chapters, you will build seven separate Joomla! websites!

## How to Use This Book

You can use this book in several ways. You can start at the beginning and go chapter by chapter, as you develop your own site. The book is carefully laid out so that introductory ideas in the earlier chapters are developed and built on to help you understand more advanced concepts later on. You can also use the book as a reference through the index, jumping to a particular aspect you need help with. Finally, the last three chapters are step-by-step guides to complete site creation based on a particular industry.

### Chapter 1: Content Management Systems and an Introduction to Joomla!

In today's fast-moving web, if you have a website that doesn't have rich functionality or fresh content, you will find yourself at a disadvantage to those who do. The idea of powering websites with a CMS has been around for some time, but it is only recently—with the advent of high quality open source CMS scripts like Joomla!—that we have seen these powerful CMS tools coming into the hands of developers like you and me.

In this chapter, I explain in detail the difference between a "traditional" website and one using a CMS. We also look at the history of Joomla! and an overview of some of its features.

### Chapter 2: Downloading and Installing Joomla!

Joomla! is one of the most popular open source CMSs on the planet. The first step in becoming part of the "Joomla!sphere," the vibrant community that exists around the Joomla! Project, is to download Joomla! and install it on your web server.

This chapter shows you how to get up and running with a Joomla! site. The two steps are to download the latest files and to install them on a web server. This chapter describes both a local installation— your home computer, to use as you read this book (if you don't have a hosting account or have a slow internet connection)—and a real web server installation.

### Chapter 3: Joomla! Administration Basics

The term "site administration" usually means the day-to-day tasks of adding content, managing users, and making sure installed components and modules are running correctly. With a properly configured Joomla! site, the administration burden is relatively low. Most of the effort can be dedicated to generating that all-important content.

In this chapter, we go on a whirlwind tour of the core administrative functions you need. I won't be step-by-step explaining every last button in the admin backend but rather picking out key functions, tips, and tricks that you need to know to keep your site humming.

### Chapter 4: Content Is King: Organizing Your Content

As a CMS, Joomla!'s primary function is to organize and present the content in your site. It does this through content articles. These discrete pieces of content must be organized into a two-level hierarchy called sections and categories.

This chapter provides an in-depth tutorial explaining how Joomla! displays its content articles and how you can organize the hierarchical structure of them. It details how to plan and organize the content and user experience for the site. It also explains how to best structure content into sections and categories for small and large sites.

### Chapter 5: Creating Menus and Navigation

Menus are perhaps the core of a Joomla! site. In a static HTML site, they merely serve as navigation. In a Joomla! site, they serve that purpose, but they also determine the layout of how a dynamic page looks and what content appears on that page when you navigate to it. The relationship between menus, menu items, pages, and modules is perhaps one of the most confusing in Joomla!. This chapter explains the relationship so that you can create a navigation scheme that works for your site.

Chapter 5 examines how the navigation (menus and links) is built for a Joomla! website and how the different aspects interact to produce a coherent navigation structure.

### Chapter 6: Extending Joomla!

It's hard to find a Joomla! powered website that has not added functionality beyond the basics with some sort of extension. The word *extension* collectively describes components, modules, plugins, and languages. There are many hundreds available free and commercially from third-party providers.

In this chapter, we look at some examples of core and third-party Joomla! extensions. We also examine how they are installed and managed in Joomla!.

### Chapter 7: Expanding Your Content: Articles and Editors

There are two main ways to add and manage content in a Joomla! site: through the frontend or through the backend. Part of the biggest attraction of Joomla! is the capability to easily add and edit content through a "what you see is what you get" (WYSIWYG) editor.

In this chapter, we begin by looking at WYSIWYG and how it functions in the backend with managers, administrators, and super administrators. We then examine how authors, editors, and publishers manage content through the frontend.

### Chapter 8: Getting Traffic to Your Site

Search engine optimization (SEO) might be one of the most maligned subjects on the web. From black hat SEO—people who use unethical methods to gain rank in search engines—to their counterparts white hat SEO—the good guys—how best to get traffic to your site is loaded with opinion and myth.

Trying to learn about SEO is difficult, to say the least. In this chapter, I emphasize *search engine marketing* (SEM). I point out some obvious SEO tips and how they apply to Joomla!, but I'll also discuss a more holistic marketing plan that includes such strategies as pay per click and blogging.

### Chapter 9: Creating Pure CSS Templates

In this chapter, we go through the steps of creating a Joomla! template. Specifically, we create a template that uses Cascading Style Sheets (CSS) to produce a layout without using tables. This is desirable because it means the template code is easier to validate to World Wide Web Consortium (W3C) standards. It also tends to load faster, is easier to maintain, and performs better in search engines. We discuss these issues in detail later in the chapter.

### Chapter 10: Creating a School Site with Joomla!

School websites tend to be medium to large in size. Two of Joomla!'s defining characteristics are its power and flexibility, but it can be time intensive to set up. This leads us to this chapter, an extensive guide to creating and setting up a school website using the Joomla! CMS.

### Chapter 11: Creating a Restaurant Site with Joomla!

The chapter looks at how to build a small business website, in this case a restaurant site, from scratch. Starting from an analysis of needs, this chapter shows how to organize possible content all the way through to adding photos and considering further extensions.

### Chapter 12: Creating a Blog with Joomla!

It seems like everyone has a blog these days. Many people still think of blogs as personal diaries, but more and more organizations and companies are using blogs to shape perceptions of who they are and what they do. Chances are, if you go to a website today, you will find a link to the owner's blog somewhere on the site. Even more common now is a section of the site dedicated to the blog.

This chapter talks about blogs in a more general sense: a dynamic communication medium for a person or organization to interact with stakeholders. We look at creating a blog site from scratch using Joomla!.

### Appendix A: Getting Help with Joomla!

Stuck with Joomla!? A tremendous amount of information is available on the web, in addition to many active communities to ask for help.

### Appendix B: Joomla! Case Studies

This appendix explains six real sites that are using Joomla!, taken from a wide range of industries and types of sites.

### Appendix C: A Quick Start to SEO

Need some quick tips to help your search engine ranking? Implement the tips included in this appendix.

### Appendix D: Installing WampServer

A quick guide to installing WampServer on your home computer. This package is important so you can follow along with all the site examples in the book.

## What Is a Content Management System?

A CMS is a collection of scripts that separates content from its presentation. Its main features are the ease of creation and editing of content and dynamic web pages. CMSs are usually very sophisticated and can have newsfeeds, forums, and online stores. They are also easily edited. More and more websites are moving toward being powered by CMSs.

Most CMSs are expensive—in the range of $50,000 to $300,000—but an increasing number of open source alternatives are becoming available. Open source CMSs have become increasingly more reliable and are now being used for important projects in many companies, nonprofits, and other organizations.

A CMS separates out the responsibilities involved in developing a website. A web designer can be concerned with the design, and non-technical people can be responsible for the content.

A modern CMS is usually defined by its capability to manage and publish content. Most CMSs do far more, taking advantage of a wide range of extensions and add-ons that add functionality.

## What Is Open Source Software?

Joomla! is an example of open source software; its nonprofit copyright holder is Open Source Matters (see www.opensourcematters.org). An open source project is developed by a community of developers around the world, all volunteering their time. Some examples of open source software you might have heard of are Firefox, Apache, Wiki, Linux, and OpenOffice. All these projects have challenged and even surpassed their commercial equivalents. If you are curious about how and why people should create powerful software for free, look for more information on these sites:

- en.wikipedia.org/wiki/Open_source
- www.opensource.org

## Things to Look For

The following are specific elements to look for when reading:

**TIP**
The tip boxes give more advanced ideas about an aspect of Joomla!. You can usually find more details at compassdesigns.net about the tip.

**NOTE**
The notes box denotes a caution about an aspect of the topic. It won't always be applicable to all situations, but you should check whether it would apply to yours. Important Notes are also used.

**THE LEAST YOU NEED TO KNOW**
The key critical concepts explained can be found in the LYNTK boxes. These are worth circling in a big red pen or writing out on a cheat sheet.

**CAUTION**
Cautions provide critical information.

Whenever a line of code is too long to fit on the printed page, it has been broken manually and the continuation preceded with a code-continuation arrow ➥. For example:

```
  <meta http-equiv="content-type" content="text/html;
charset=utf-8" />
➥<meta name="robots" content="index, follow" />
```

Placeholders in code are italic mono. For example, in the following code, *location* and *option* should be replaced with your specific information:

```
<jdoc:include type="modules" name="location" style="option" />
```

## Joomla!

The full and proper name of the Joomla! CMS includes an exclamation point, as shown here. For the sake of readability, and a tree or too, I've kept the exclamation point in heads but dropped it throughout the chapter text.

## www.joomlabook.com

You can find more information about this book, including complete browsable and downloadable versions of all the sites created in the chapters, at www.joomlabook.com.

## Writing About Open Source Products

As with many open source products, Joomla! changes on a very short release cycle. New versions with slight changes can be released in as little as six weeks, and usually the changes are difficult to find out about. This makes writing for open source challenging. If you find minor inconsistencies in this book, they're most likely caused by these minor updates. To stay informed of recent changes to Joomla!, read the discussions of Joomla! versions in the forum at www.joomlabook.com.

Chapter 9

# Creating Pure CSS Templates

## In This Chapter

This chapter walks through the steps of creating a Joomla template. Specifically, you will create a template that uses Cascading Style Sheets (CSS) to produce a layout—without using tables. This is a desirable method because it makes the template code easier to validate to World Wide Web Consortium (W3C) standards. It also tends to load faster, is easier to maintain, and performs better in search engines. These issues are discussed in detail later in the lesson. This lesson covers the following topics:

- What is a Joomla template? What functions does a Joomla template perform, and what is the difference between a template that has no content and a template whose content is added to the CMS?
- How does the localhost design process differ from that of a static HTML or XHTML web design process?
- What are the implications of tableless design in Joomla, and what is 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 three-column layout by using CSS rather than tables?

- What are the basic CSS styles that should be used with Joomla, and what are the default styles that the Joomla core uses?
- How do you place and style modules, and what are some new techniques for rounded corners?
- What would be a simple strategy for producing lean CSS menus that mimic the effects of menus developed with JavaScript?
- How do you control when columns are shown and hide them when no content is present?
- What are the proper steps in creating a Joomla 1.5 template?

**A DISCLAIMER OR TWO OR THREE**

This is probably the most technical chapter in the book. To be successful with this chapter, you need a firm grasp of XHTML and CSS; for example, you need to understand what *float* does and how to clear it.

If you are not sure if you have the skills needed to make a Joomla template, I strongly advise grabbing a template from Joomlashack.com. Yes, I am recommending a template company that I am involved in, but at Joomlashack we specialize in creating bloat-free simple templates that are easy to understand and customize. A good way to learn is to grab one of our free templates and try to reverse engineer it to see how it works.

## What Is a Joomla! Template?

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

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

Fig. A—Joomla Template with Sample Content

Fig. B—Joomla Template with Little or No Content


FIGURE 9.1 A template with and without content.

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:

- Joomla does all the work of placing content within pages. You can add new information to existing blog pages simply by typing a new article. The template and its CSS make sure it appears stylistically consistent with other content on the site.
- 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 determining whether a site meets modern web standards. In a standards-compliant site, the HTML tags for tables are reserved for presenting tabular data and not laying out a page into columns.
- You can apply a new template, and hence a completely new look to a website, instantly. This can involve different locations for positioning content and modules, as well as colors and graphics.

**THE LEAST YOU NEED TO KNOW**

Modern websites separate content from presentation by using templates and CSS. In Joomla, a template controls the presentation of content.

## The Localhost Design Process

The web page you see at a Joomla-powered website is not static; it is generated dynamically from content stored in the database. When content in the database is changed,

all pages that display that content are instantly changed. The page you see is created through various PHP commands in the template that query the database. Because the template looks like lines of code instead of content, it 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, so you don't need to code the HTML. However, using such an editor is not possible in the Joomla template design process because WYSIWYG editors cannot display and edit dynamic pages. Therefore, you must code a template and its CSS manually and view the output page from the PHP on a served page that you frequently refresh as you make changes. With a fast enough connection, this could be a web server, but most designers use a local server, or localhost, on their own computer—a piece of software that serves the web pages on your computer, such as the localhost setups described in Chapter 2, "Downloading and Installing Joomla!"

There is no "right way" to create a web page; how you do it depends on your background. Those who are more graphics inclined tend to make an "image" of a page in a graphics program such as Photoshop and then break up the images so that they can be used for the web (known as *slicing and dicing*). More technology-based designers often jump straight into the CSS and start coding fonts, borders, and backgrounds. However, as just mentioned, as a Joomla template designer, you're limited by the fact that you cannot instantly see the effect of your coding in the same editor. You can therefore use the following modified design process:

1. Have a localhost server loaded with content running in the background to "run" Joomla.
2. Make your edits to the HTML and CSS with an editor and then save your changes to the server.
3. View the pages affected by your edits in a web browser.
4. Return to step 2.

**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 resulting pages to check them.

## Localhost Server Options

In Chapter 2, you saw how to install a web server (WampServer) that will run on your computer. To move further along in this chapter, you need to have WampServer installed. If you haven't done so yet, go ahead and install it. I'll wait right here.

**TIP**

One useful technique for making the design process more efficient is to serve a web page that you are designing and then copy and paste the generated source from your browser into an editor. For example, once the CSS for your layout is set up, you can use a localhost server to serve a page, and then you can view the source of the page. You can then copy and paste the source code into your editor, and then you can easily style the page using CSS, without having to go through the cycle of steps described earlier. When you have completed your editing, you can copy your perfected CSS styles back to the server.

On a hosted web server, you can edit the HTML template and CSS files in the backend while having the frontend open in another tab of your browser. As you save your changes, you can simply refresh the frontend view in order to see the impact.

With a localhost setup, you have the added convenience of direct access to the files to edit them with the editor of your choice. As you save your changes, without having to close the editor, you can refresh the frontend view in your browser and see the impact.

**A FREE XHTML EDITOR**

In addition to commercial editors, such as Dreamweaver, some free editors are available. Nvu is a solid choice that has built-in validation and is 100% open source. This means anyone is welcome to download Nvu at no charge (net2.com/nvu/download.html). You can even download the source code and make special changes, if desired.

**TIP**

When using Firefox as you're designing a template, you can use three add-in tools that are of particular help: the Web Developer toolbar, Firebug, and ColorZilla.

## 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 typically does so for all three (see Figure 9.2). The easiest way to achieve these three goals is to use the framework laid out in the W3C web standards.

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

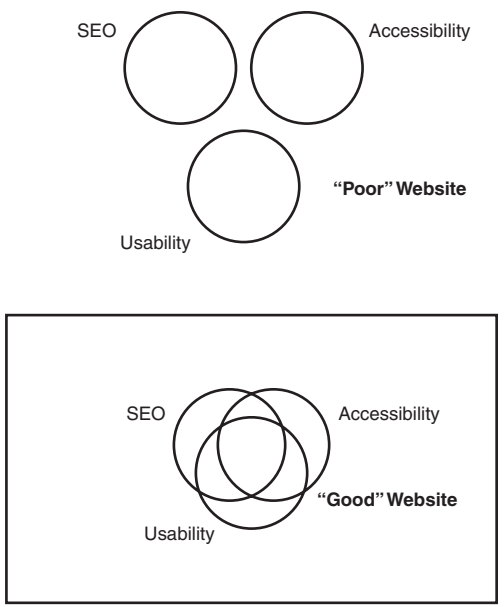


FIGURE 9.2 The overlap between usability, accessibility, and SEO.

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 W3C, whose director, Tim Berners-Lee, is credited with inventing the web in 1989.

To 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 was born. Each generation has introduced new features, and the manufacturers have come up with different, sometimes proprietary, tags (names) for those features. Each browser has tended to have a different syntax, or "dialect," and quirks for implementing the same base HTML language. New browsers have appeared, and some old ones have disappeared (remember Netscape?).

Current W3C standards serve to (hopefully) push manufacturers to release more compliant browsers that read the same language and display pages more consistently so that designers can design to a single common platform.

Another complicating factor is that historically, different browser makers (such as Microsoft) tend to have their browsers interpret HTML/XHTML in slightly different ways. This has led to web designers having to design their websites to support older browsers rather than new ones. Designers and website owners have often decided that it's important that a web page appear properly in 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 good foundation for making itself accessible, usable, and optimized for search engines. 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 (just make sure you use the correct `DOCTYPE` when you try to validate your code; see www.compassdesigns.net/tutorials/82-installing-joomla-doctype-and-the-blank-joomla-template.html). At its simplest, a site that meets W3C validation is likely to also use semantic HTML or XHTML and separate its content from presentation by using CSS.

Ask five designers what web standards are, and you will get five different answers. But most agree that web standards are based on using valid code, whether HTML or XHTML (or others), in the manner specified in the latest version of the standards.

### Semantically Correct Code

As mentioned earlier, being semantically correct means that the HTML or XHTML tags in a web page describe only content, not presentation. In particular, this means structured organization of H1 tags, H2 tags, and so on and using tables only for tabular data, not for layout. One area where Joomla template designers compromise slightly on being purely semantically correct is the convention of naming the left and right columns of a two- or three-column layout as, well, `left` and `right` instead of the more semantically correct `sidebar` or `sidecolumn`. If these are only position names used in the template's PHP, they are technically correct. If they are also used to define matching

classes in the HTML and CSS, it's a forgivable convenience to have everything associated with displaying the page's left column named or classed as `left`. In the examples that follow, you will see that the position of `left` is styled with the class `sidebar` and `right` is `sidebar-2`, which is semantically correct code.

### Cascading Style Sheets (CSS)

Closely related to making code semantically correct is using CSS to control the look and layout of a web page. CSS is a simple mechanism for adding style (for example, fonts, colors, spacing) to web documents (see www.w3.org/Style/CSS/*).* CSS exist parallel to the HTML and XHTML code and let you completely separate content (code) from presentation (CSS). To see this in action, check out CSS Zen Garden (www.csszengarden.com), a site where the same XHTML content is displayed in different and unique ways, just by changing the CSS file. The resulting pages look very different but have exactly the same core content.

Designing Joomla-powered sites currently presents considerable challenges in terms of meeting validation standards. In the first series of Joomla releases, 1.0.X, the code used a significant number 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 many third-party developers of components and modules are still using tables to generate their layouts.

Fortunately, the Joomla core development team recognized this issue with Joomla. In Joomla 1.5, 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—in whatever way they want.

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: CSSTemplateTutorialStep1

To understand the contents of a template, let's start by looking at a blank Joomla template.

**NOTE**
There are two ways you can use this chapter. You can start with new files and type in the code shown here to slowly build the template. This process is time-consuming and prone to error. Instead, you can refer to the supplied templates from www.joomlabook.com. There are four templates, each of which corresponds to the stage of its development at the *end* of the related section in this chapter. Download the sample template that matches the section you are reading, and you can follow along.

You can also follow along by installing these four templates in your localhost, in which case you'll be able to see your edits and tests live on the frontend.

## Template File Components

This section reviews the manual process of setting up template files. Normally, you would install the template using the Joomla installer, which takes care of all these steps.

When constructing your own templates, you need to set up several files and folders in a coordinated manner. A template needs to contain various files and folders. These files must be placed in the `/templates/` directory of a Joomla installation, each in a folder designated for that template. If you had two templates installed called Element and Voodoo, your directory would look something like this:

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

Note that the directory name for a template must be the same as the name of the template—in this case, `element` and `voodoo`. These names are case-sensitive and shouldn't contain spaces.

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

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

These filenames and locations must match exactly because this is how 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 it loads 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 unpacking and installing a template when using the extension installer in the administrative backend.

The second key file is the primary template file that generates pages, `index.php`. This file is the most important in a Joomla template. 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 HTML/XHTML.

Almost all templates use additional files. It is conventional (although not required by the Joomla 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 lists the files commonly found in a template.

TABLE 9.1 Core Files Needed for a CSS-Based Template

| /templatename/folder/filename | Description |
|---|---|
| `/element/template_thumbnail.png` | A web browser screenshot of the template (usually reduced to around 140 pixels wide by 90 pixels high). After the template has been installed, this functions as a preview image that is visible in the Joomla administration Template Manager and also the template selector module in the frontend (if used). |
| `/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 you will see later that there are advantages to having other CSS files, too. |
| `/element/images/logo.png` | Any images that go with the template. Again for organization reasons, most designers put them in an images folder. Here we have an image file called `logo.png` as an example. |

### `templateDetails.xml`

The `templateDetails.xml` file acts as a manifest, or packing list, that includes a list of all the files or folders that are part of the template. It also includes information such as the author and copyright. Some of these details are shown in the administrative backend in the Template Manager. An example of an XML file is shown here:

**NOTE**
If you are following along and creating the template as you read, at this point, you should open up a text editor, create a file called `templateDetails.xml`, and make sure it includes the code 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>favicon.ico</filename>
            <folder>css/</folder>
            <folder>images/</folder>
            <folder>js/</folder>
       </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 look at 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 you are installing a template and that it is for Joomla 1.5.
- **`<name>TemplateTutorial15</name>`**—This line 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, such as spaces. If you're installing manually, you need to create a directory whose name is identical to the template name.
- **`<creationDate>August 2007</creationDate>`**—This is 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>`**—This is the name of the author of this template—most likely your name.
- **`<copyright>GPL</copyright>`**—Any copyright information goes in this element.
- **`<authorEmail>compassdesigns@gmail.com</authorEmail>`**—This is the email address at which the author of this template can be reached.
- **`<authorUrl>www.compassdesigns.net</authorUrl>`**—This is the URL of the author's website.
- **`<version>1.0</version>`**—This is the version of the template.
- **`<files></files>`**—This is a list of various files used in the template. The files used in the template are laid out with `<filename>` and `<folder>` tags, like this:

```
<files>
     <filename>index.php</filename>
     <filename>templateDetails.xml</filename>
     <filename>favicon.ico</filename>
 <folder>css/</folder>
 <folder>images/</folder>
 <folder>js/</folder>
</files>
```

The "files" sections contain all generic files, such as 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>` tags. You can also include whole folders, such as an image folder, by using the `<folder>` tag.

- **`<positions></positions>`**—This shows the module positions available in the template. It is the list of page locations, such as `top`, `left`, and `right`, defined in the template in which modules can be set to appear, using the Position drop-down of the Module Manager. The position names in this list must precisely match the PHP code that generates content for each listed position inside `index.php`.
- **`<params></params>`**—This section describes the parameters that can be set in the backend and passed as global variables to allow advanced template functions, such as changing the color scheme of the template.

### `index.php`

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

**NOTE**
If you are following along and creating the template as you read, at this point you should open a text editor, create a file called `index.php`, and make sure it includes the following code excerpt.

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 the 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 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—how various HTML tags are handled and, more importantly, how the browser interprets CSS. The following observation from alistapart.com should give you further understanding:

> [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 She's Got Mail.

You can use several DOCTYPEs. Basically, the DOCTYPE tells the browser what version of HTML was used to design the page, whether it has some legacy code or also contains XML, and therefore how to interpret the page. Here the words *strict* and *transitional* start getting floated around (`float:left` and `float:right` usually) to indicate whether legacy code was included. Essentially, ever since the web started, different browsers have had different levels of support for various HTML tags and versions of CSS. For example, Internet Explorer won't understand the `min-width` command to set a minimum page width. To duplicate an effect so that it displays the same in all browsers, you sometimes have to use browser-specific "hacks" in the CSS that make up for shortcomings in each browser's adherence to the published standards.

*Strict* means the HTML (or XHTML) will be interpreted exactly as dictated by standards. A *transitional* DOCTYPE means that the page will be allowed a few agreed-upon differences from the standards (for example, continued use of discontinued tags).

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 backward compatible, so Internet Explorer 6, for example, will render the page as if it were Internet Explorer 4.

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`, which is a relative link on the WC3 server. You need the full path, as shown earlier.
- Microsoft set up Internet Explorer 6 so you could have valid pages but be in quirks mode. This happens when you have an `xml declaration` put before instead of after 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 information I just gave you about Internet Explorer 6 quirks mode is important. In this chapter, you're designing only for Internet Explorer 6 and later, and you need to make sure that it's running in standards mode to minimize the hacks you have to do later on.

**NOTE**

Making a page standards compliant, so that you see `valid xhtml` at the bottom of the page, does not require really difficult coding or hard-to-understand tags. It merely means that the code you use follows the rules—it 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 that 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

Let's look at the structure of the `index.php` file header; you want it to be as minimal as possible but still have enough for a production site. The header information you 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 this?

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

The next line is for including more header information:

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

This code snippet inserts in the generated page all of the header information that is set in the Global Configuration. In a default installation, it includes the tags shown here:

```
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  ➥<meta name="robots" content="index, follow" />
  <meta name="keywords" content="joomla, Joomla" />
  <meta name="title" content="Home Page" />
  <meta name="author" content="Administrator" />
  <meta name="description" content="Joomla! - the dynamic portal engine
  ➥ and content management system" />
  <meta name="generator" content="Joomla! 1.5 - Open Source Content
  ➥ Management" />
  <title>Home Page</title>
  <link href="/templates/yourtemplate/favicon.ico" rel="shortcut icon"
type="image/x-icon" />


  <script type="text/javascript" src="http://localhost//yoursite/
➥media/system/js/mootools.js"></script>
  <script type="text/javascript" src="http://localhost//yoursite/
➥media/system/js/caption.js"></script>
```

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

The last lines in the header provide links to CSS files for Joomla-generated pages in general and also in this 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 ?>` returns the name of the current template. Writing it in this way rather than writing the actual path makes the code more generic. When you create a new template, you can just copy this line (along with the whole header code) and not worry about editing anything.

The template CSS can include any number of files, such as conditional ones for different browsers and for different media, such as print. Adding the following code detects and adds an additional CSS file that targets the quirks of Internet Explorer 6:

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

The next example is part of a technique for using a template parameter. In this case, a color scheme selected as a parameter in the Template Manager is loading a CSS file that has the same name as the selected color:

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

It might generate this:

```
<link rel="stylesheet" href="templates/voodoo/css/blue.css"
➥type="text/css" />
```

### The Joomla! Page Body

Now that the `<head>` part of the page is set up, we can move on to the `<body>` tag. Creating your first template will be easy! Ready?

To create the template, all you need to do is use Joomla statements that insert the contents of the mainbody, plus any modules you want:

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

The template contains the following, in reasonably logical viewer order:

- The name of the site
- The top modules
- The left modules
- A breadcrumb bar
- The main content
- The right modules
- The footer modules

At this point (if you preview it), the site does not look very awe inspiring (see Figure 9.3).


FIGURE 9.3 An unstyled template.

**THE LEAST YOU NEED TO KNOW**

The most basic template simply displays the Joomla modules and mainbody (component). In a CSS-based template, layout and design are accomplished by the CSS, not by the template.

You want to come as close to semantic markup as possible. From a web point of view, this 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 you have with your template so far is really only the *potential* for semantic layout. If you were to go ahead and put random modules in random locations, you would have a mess. An important consideration for CMS sites is that a template is only as good as the population of the content. This often trips up designers who are trying to validate their sites.

Notice that you use the first of a number of commands specific to Joomla to create this output:

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

The PHP echo statement simply outputs a string from the `configuration.php` file. Here, you use the site name; you could as easily use 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 XHTML output, from either modules or components.

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

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

**NOTE**
Interestingly, you can insert multiple instances of component output. I'm not sure why you would want to, but you can!

This line inserts the output for a module location:

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

This line generates content for all modules that have their position set to `right`. The content generated for those modules is placed in the page in the order set in the Order column of the Module Manager. This is the full syntax:

```
<jdoc:include type="modules" name="location" style="option" />
```

We'll look at the various options for styles in the section "Modules in Templates," later in this chapter.

**CSSTEMPLATETUTORIALSTEP1**
At this point, you have a very bare template.

I have created an installable template that is available from www.joomlabook.com: `CSSTemplateTutorialStep1.zip`. By opening this file, you can install a template that has only two files, `index.php` and `templateDetails.xml`. In these files, I removed references to other files to give barebones output with no CSS. This is 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: CSSTemplateTutorialStep2

In this section, you will use pure CSS to make a three-column layout for the Joomla template. You will also be making it a "jello" layout. There are three main types of web page layouts: fixed, fluid, and jello—and they all refer to how the width of the page is controlled. A fixed layout has the width set to some fixed value. A fluid layout can grow and shrink to the browser window, and a jello layout is fluid but between some minimum and maximum values.

The width of the page is an issue because of the many browser resolutions at which people surf the web. The majority, 79%, are using 1024×768 and higher (see www.upsdell.com/BrowserNews/stat_trends.htm#res). Only a couple years ago, fluid layout

was preferable for maximum flexibility. Now, however, many users have big (2,000+-pixel) screens. A fluid layout becomes unreadable because the human eye cannot correctly scan lines over 960 pixels. Thus, I now prefer to use the jello layout.

A typical design might use tables to lay out the page. Tables are useful as a quick solution in that you just have to set the width of the columns as percentages. However, tables also 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 "spacer GIFs," which are 1x1 transparent images placed in each cell of the table to keep the cells from collapsing. Even big companies sometimes fall into the table trap.

There are a couple major problems with a site using tables for layout:

- They are difficult to maintain. To change something, you have to figure out what all the table tags, such as `<tr>` and `<td>`, 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 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 three-column layout) before they get to the middle column, where the important stuff is located. 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 by using CSS. For a quick introduction, a good source is Brainjar's "CSS Positioning" (see www.brainjar.com/css/positioning/).

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

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

In this chapter, you will use floats to position your content. At its most basic, your template might look as shown in Figure 9.4. It's still not very exciting, but let's look at what the different parts are all about.

The CSS styles are defined in the head of the file to show what is going on, but normally they are contained in an external file that the page links to, such as the file `template.css` mentioned earlier in this chapter.


FIGURE 9.4 Basic template layout.

In Figure 9.4, each column—left, middle, and right—is given its own element. Each is floated left and given a percentage 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 you build your second template in this chapter, you will have to use a more advanced clearing technique.)

To improve the layout and to add some breathing room to the content, you need to add some column spacing, commonly called *gutter*. Unfortunately, there is a problem here. You might know that Internet Explorer generally does not interpret CSS correctly. One problem is that it calculates width in a unique way. You can cope with this problem by not using any padding or borders on something that has width. To get the gutter, you instead add another, narrower, `<div>` element that fits inside the columns.

To the CSS you add this:

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

The resulting `<body>` code for `index.php` is as follows:

```
<body>
<div id="wrap">
  <div id="header">
    <div class="inside">
        <?php echo $mainframe->getCfg('sitename');?>
      <jdoc:include type="modules" name="top" />
    </div>
  </div><!--end of header-->
  <div id="sidebar">
    <div class="inside">
      <jdoc:include type="modules" name="left" />
    </div>
  </div><!--end of sidebar-->
  <div id="content">
    <div class="inside">
      <jdoc:include type="component" />
    </div>
  </div><!--end of mainbody content-->
  <div id="sidebar-2">
    <div class="inside">
      <jdoc:include type="modules" name="right" />
    </div>
  </div><!--end of sidebar-2-->
  <div id="footer">
```

```
    <div class="inside">
Powered by <a href="http://joomla.org">Joomla!</a>. Valid <a
➥href="http://validator.w3.org/check/referer">XHTML</a> and <a
➥href="http://jigsaw.w3.org/css-validator/check/referer">CSS</a>.
    </div>
  </div><!--end of footer-->
</div><!--end of wrap-->
</body>
```

The `template.css` file looks like this:

```
/*Compass Design template.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;}
```

**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 the following:

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

can all be replaced with this:

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

> There are shorthand styles at the beginning of each style definition. After you have figured out the styles, you can fill in the shorthand versions and delete the long versions.
>
> For example, the shorthand syntax for `margin` is as follows:
>
> ```
> margin: margin-top | margin-right | margin-bottom | margin-left
> ```
>
> As another example, this is the shorthand syntax for `font`:
>
> ```
> font: font-size |font-style | font-variant | font-weight |
> ➥line-height | font-family
> ```
>
> Rather than use this:
>
> ```
> font-size:1em; font-family:Arial,Helvetica,sans-serif;
> ➥font-style:italic; font-weight:bold; line-height:1.3em;
> ```
>
> you can use this:
>
> ```
> font:bold 1em/1.3em Arial,Helvetica,sans-serif italic;
> ```
>
> You can read more about this syntax at 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. In essence, it says to fit the page into a space that's 760 to 960 pixels wide; display the header full width and then divide the space into a column that's 20% of this total width followed by a column that's 60% of the width, followed by a final one that's 20% of the width. After that, it is to display the footer at full with. For the content that fits inside each of these spaces, give it 10 pixels of padding.

However, this simple layout is viewer ordered in the sequence in which you see content on the screen and not source ordered to place the most important content at the beginning of the generated HTML source yet still have the same viewer-ordered appearance onscreen, with the left column displayed before (that is, to the left of) the center column. For that, you must use a more advanced layout known as a *nested float*.

Source-ordered layouts perform better for SEO than do layouts where the important content occurs late in the code. From a Joomla site perspective, the important content is that which comes from the mainbody component. For now, to keep the CSS simple, we'll stick with this viewer-ordered layout, and we'll change to source-ordered layout later in the chapter.

## Default CSS

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

```
/* template.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;}
```

Here, you center the page by using a small hack. You have to do this because Internet Explorer does not interpret CSS accurately. With a standards-compliant browser, you could just use `margin:0` 10%; to center the page or `margin:auto`; but Internet Explorer does not recognize that, so you center the "text" of the whole page as a way of centering the "wrap" `<div>`, and then align the text back to the left when you are inside the wrap `<div>`.

In celebration of Internet Explorer 7's support of min/max width, you can add minimum and maximum widths. Note that you have to add a tiny hack for Internet Explorer 6, which does not understand these settings—it will just ignore the whole `!important` statement line and have a plain fixed 960-pixel width.

**NOTE**

It might seem strange to define 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 you add width changer buttons, you need to change only one value.
- You still have a maximum width so why not go all fluid? Many viewers on the web now have enormous screens. Usability research says that lines of text over 960 pixels wide are difficult to read because the eyes have to go a long way to go to the next line. Limiting the fluidity makes the site more usable and accessible.

The true typography rule is to limit the maximum width of a column of text to the equivalent of about 60 characters. There is a lot of discussion on the web about defining widths in ems so that if the font is set larger, the width of the column might expand, with the font beyond 580 pixels for the center column (60% of 960 pixels, less the 10-pixel padding). A Google search of "fluid v. liquid v. elastic CSS layouts" will inundate you with typographic esoterica. To keep it simple, we'll use 960 pixels for the full body width and set a font size for the center column that fits well in 580 pixels.

You add a new style to the columns, `overflow:hidden`, that makes the page "break" more consistently as you reduce its width.

As you begin working on typography with CSS, you should set some overall styles and include 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;
   }
```

The purpose of a global reset is to override the default settings that are different in every browser and get to a clean, consistent starting point, regardless of which browser the page is being displayed on. Everything is given a zero margin and padding, and then all block-level elements are given a bottom and a bottom margin. This helps achieve browser consistency. (The first CSS selector above is called the *star selector*, and it acts as a universal selector even in Internet Explorer 6.) You can read more about the global reset at www.clagnut.com/blog/1287/ and www.leftjustified.net/journal/2004/10/19/global-ws-reset/. You set the font size to 76% to try to get more consistent font sizes across browsers. All font sizes are then set in ems. Setting `line-height:1.3` helps readability. When you set fonts and line heights in ems, the pages are more accessible because the viewers will be able to resize the fonts to their own preferences, and the pages will reflow and remain readable. This is discussed further at www.thenoodleincident.com/tutorials/typography/template.html.

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

Notice that the side columns do not reach the footer. This is because they extend only as far as their content; where the space is white on the left and on the right, the side columns don't exist. If you have a template that has a white background for all three columns, this is no problem. You will use this approach and will have boxes around the modules. If you want equal-height columns that are colored or have boxes, you have to use a background image that tiles vertically. This technique, called *faux columns*, is described at www.stopdesign.com/log/2004/09/03/liquid-bleach.html and www.meyerweb.com/eric/thoughts/2004/09/03/sliding-faux-columns/.

## Joomla-Specific CSS

Although Joomla 1.5 has the functionality to override the core output in a template, its default rendering still uses a significant number of tables to output content in the mainbody. Along with these tables, CSS class assignments are available for styling pages. Based on some research by various community members, Table 9.2 shows the current list of CSS style classes. Note that it does not include generic web page styles, such as `H1`, `H2`, `p`, `ul`, `a`, `form`, and so on.


FIGURE 9.5 A basic template with typography.

TABLE 9.2 Default CSS Style Classes in Joomla 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 |
| contentpagetitlw | | |

Many style classes in Table 9.2 actually have comparable CSS styles that are more specific in their definitions. Basically, a more specific rule overrides a less specific rule. Here's an example:

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

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

The color of a link and the color of `.contentheading` will be *red*, as that rule is more specific (because `.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 the module. `table.moduletable` will apply the style only to a table with `class="moduletable"` on it. `.moduletable` will apply the style regardless of what element the class is on.

Consider these examples:

```
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; the following websites 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, although your template's `index.php` file has no table tags within it, the generated page includes several tables. This is because Joomla 1.5 has a new feature called *template overrides* (more about them later) that eliminate tables from layout; however, to remain backward compatible with template designers, Joomla will output them as the default. As mentioned earlier, this slows down the pages and makes them more difficult to update. To reduce the number of tables, when you call the modules, you need to use style parameters in the `jdoc:include` statements.

**THE LEAST YOU NEED TO KNOW**
Joomla will output specific elements, including IDs and classes, in the code of a web page. These can be predicted and used to style the design using CSS.

## Modules in Templates

When a module is called in the `index.php` file, there are several options for how it is displayed. The syntax is as follows:

```
<jdoc:include type="modules" name="location" style="option" />
```

The style, which is optional, is defined in `templates/system/html/modules.php`. Currently, the default `modules.php` file contains the following layout options: `table`, `horz`, `xhtml`, `rounded`, and `none`. Let's take a brief glimpse at the lines of code needed for each of these options:

`OPTION="table"` (default display) modules are displayed in a column. The following shows the output from Joomla when we use the `"table"` option. Note the PHP statements would be replaced by actual content:

```
<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 horizontally. Each module is output in the cell of a wrapper table. The following shows the output from Joomla when we use the `"horz"` option:

```
<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 simple `div` elements, with the title in an `H3` tag. The following shows the output from Joomla when we use the `"xhtml"` option:

```
<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 `$style` is used, the name of the `<div>` changes from `moduletable` to `module`. The following shows the output from Joomla when we use the `"rounded"` option:

```
<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 (`xhtml` and `rounded`) are much leaner in code, which makes it easier to style the web pages. I don't recommend using the options (suffixes) `table` (default) or `horz` unless absolutely needed.

If you examine the `modules.php` file shown earlier, you will see all these options that exist for modules. It's easy to add your own; this is part of the new templating power of Joomla 1.5. (We will look at this in more detail in the section "Template Overrides," later in this chapter.)

To develop a template, you can put the module style `xhtml` on all your modules in `index.php`:

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

**NOTE**

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

**THE LEAST YOU NEED TO KNOW**

In Joomla 1.5, you can completely customize the output of modules, or you can use the prebuilt output by setting style options for each module position. All these options are referred to as module *chrome*.

Let's remove the background from the layout `div`s and add some CSS to style the modules with a border and a background for the module titles.

The typography portion of your CSS file should now look 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*/ }
```

Here you have added specific style rules for the modules generated with `style="xhtml"` and therefore generated each with a `<div>` of class `.moduletable` and having the module's heading displayed in an `<h3>` tag within that `<div>`.

**NOTE**
Several of the menus in the default Joomla installation have the menu suffix `_menu` in the module properties. To get everything behaving properly, that parameter has been deleted in this example.

The typography CSS you've created now produces the result shown in Figure 9.6.


FIGURE 9.6 A basic template with module and title styling.

## Menus in Templates

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

Again, using CSS lists rather than tables results in reduced code and easier markup. After setting the Module Manager parameters so that all your menus render as *lists*, you have only 12 tables (you'll see how to remove the rest using the Joomla 1.5 override feature). Remember, the `list` setting is used in Joomla 1.5; `flat list` is from Joomla 1.0 and will be deprecated. 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 sample 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 has a selection of more than 30 menus, all using the same underlying code (see www.css.maxdesign.com.au/listamatic/index.htm). It's called the Listamatic. There is a slight difference in the code that you have to change in order to adapt these menus to Joomla.

These list-based menus use the following general code structure:

```
<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 href="#">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, you need to have some sort of enclosing `<div>`. You can achieve this by using module suffixes. Recall that the output of a module with `style="xhtml"` is as follows:

```
<div class="moduletable">
  <h3>...Module_Title...</h3>
  ...Module_Content...
</div>
```

If you add a module suffix called `menu`, it will get added to the `moduletable` class, like this:

```
<div class="moduletablemenu">
  <h3>...Module_Title...</h3>
  ...Module_Content...
</div>
```

So when choosing a menu from the Listamatic, you would need to replace the `navcontainer` class style in the CSS with `moduletablemenu`.

**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 that these roles might be depreciated in future versions beyond Joomla 1.5.

This use of a module class suffix is useful. 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 `list` option for menu output. You can then make use of many available free resources to obtain the CSS to display a list as a navigation menu.

For your site, say that you want to use List 10 by Mark Newhouse (see www.css.maxdesign.com.au/listamatic/vertical10.htm). Your CSS will look like this:

```
.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;
   }
```

You then need to add the module suffix `menu` (no underscore in this case) to any modules for menus you want styled using this set of CSS rules. This will produce a menu like what's shown in Figure 9.7.

**TIP**

When trying to get a particular menu to work, create a default Joomla installation and then look at the code that makes up the main menu. Copy and paste this code into an HTML editor (such as Dreamweaver). Replace all the links with `#`, and then you can add CSS rules until you achieve the effect you want. 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">
<!--
.moduletablemenu{

 ...  your menu testing css ...

}
-->
</style>
</head>
<body>
```

```
<div class="moduletablemenu">
<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 A basic template with menu styling.

## Hiding Columns

So far, you have a layout such that you always have three columns, regardless of whether there is any content positioned in those columns. From the perspective of a CMS template, this is not very useful. In a static site, the content would never change, but you want to give your site administrators the ability to put it, without having to worry about editing CSS layouts. You want to be able to turn off a column automatically or collapse it if there is no content to display there.

Joomla 1.5 provides an easy way to count the number of modules generating content for a particular position so that you can add some PHP testing of these counts and hide any empty columns or similar unused `<div>` containers and adjust the layout accordingly. This PHP `if` test syntax for modules is as follows:

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

There are four possible conditions. For example, let's count the number of modules in Figure 9.7. You could insert this code somewhere in `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');?>
```

So if we inserted this code into our template, we might get the following results with the sample Joomla content:

- **`countModules('left')`**—This returns `4` because there are four modules on the left.
- **`countModules('left and right')`**—This returns `1` because there is a module in the left and right positions. (Both tests are true (> 0), so the `and` test `true and true` is a logical true.)
- **`countModules('left or right')`**—This returns `1` because there is a module in the left or right position. (Both tests are true (> 0), so the `or` test `true or true` is a logical true.)

- **`countModules('left + right')`**—This returns `7` because it adds together the modules in the left and right positions.

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

There are several ways to do this. You 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 are there. To make it as easy as possible, you can use 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";
?>
```

The result is to set a PHP variable called `contentwidth` with a number you can use later:

- If there is nothing in `left` *or* `right`, display the center column at 100%.
- If there is something in `left` *or* `right`, display the center column at 80%.
- If there is something in `left` *and* something in `right`, display the center column at 60%.

In all three cases, the side column(s) can be styled the same, at 20% width.

You then need to change the `index.php` file in the content `div` to this:

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

Then you change `template.css` to this:

```
#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.

You can also accomplish this by having the `if` statement import a sub-CSS file.

**TIP**

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

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

You are halfway there, but now you have expanded the width of the center column to accommodate any empty (soon to be hidden) side columns.

### 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 you don't do this, the pages will have empty `<div>`s in them, which can lead to cross-browser issues.

To not generate an empty `<div>`, you use the following `if` statement:

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

When you use this code, if there is nothing published in position `left`, then `<div id="sidebar">`; also, everything within it will not be included in the generated page.

Using these techniques for the left and right columns, your `index.php` file now looks as follows:

**NOTE**

We also need to add an `include` for the `breadcrumbs` module, the module that shows the current page and pathway. Note that to have `breadcrumbs`, the code for that position needs to be included in the `index.php` file and also `breadcrumbs` 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><!--end of header-->

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

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

<?php if($this->countModules('right')) : ?>
  <div id="sidebar-2">
    <div class="inside">
      <jdoc:include type="modules" name="right" style="xhtml" />
    </div>
  </div><!--end of sidebar-2-->
  <?php endif; ?>
  <?php if($this->countModules('footer')) : ?>
  <div id="footer">
    <div class="inside">
      <jdoc:include type="modules" name="footer" style="xhtml" />
    </div>
  </div><!--end of footer-->
  <?php endif; ?>
</div><!--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. You can accomplish this by using conditional PHP statements to control whether the code for a column is generated and also that links other content to different CSS styles; you can either modify a class name or load an entire alternative CSS file.

I recommend a slightly different way of producing the footer. In the example shown here, it is hard-coded into the `index.php` file, which makes it difficult to change. Right now, the "footer" module in the administrative backend shows the Joomla copyright and can't be edited easily. It would make much more sense to have a custom HTML or XHTML module placed in a position called `bottom` so the site administrator could more easily change it. 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, you replace this:

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

with this:

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

You 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 necessarily correspond to any particular location. The location of a module is completely controlled by the template designer, as you have seen. It's customary to place a module in a location that is connected to the name, but it is not required.

The basic template created in this section shows some of the fundamental principles of creating a Joomla template.

**CSSTEMPLATETUTORIALSTEP2**

You now have a basic but functional template. Some simple typography has been added, but more importantly, you 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 you have the basics done, you can create a *slightly* more attractive template, using the techniques you have learned.

## Making a Real Joomla! 1.5 Template: CSSTemplateTutorialStep3

You need to start with a comp. A *comp*, short for *composition*, is a drawing or mockup of a proposed design that will be the basis of the template. In this section, you will be using the Bold template, kindly donated by Casey Lee, the lead designer from Joomlashack (www.joomlashack.com), and you can see it in Figure 9.8.


FIGURE 9.8 A design comp from Joomlashack.

## Slicing and Dicing

The next step in the process is slicing. You need to use your 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 design—than Photoshop.)

This process is probably a whole book in itself. To get a sense of how to slice up a design, you can look at the source PNG file in Fireworks for the template and you'll be able to see the slices.

## Setting Up Module Locations

The Bold 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, you need to designate the following positions:

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

Nothing else should be published in these locations. (Quickly scan position assignments in the Module Manager and reposition or disable modules as necessary.)

## Header

The header image has a faint swish at the top. We put the image in as an untiled background and then assign a matching fill color behind it. That way, the header can scale vertically if you need it to (for example, if the fonts are resized). You also need to change the color of any type to white so that it will show up on the black background.

You will also use another background image for the search box. You need to make sure to target the correct input by using CSS specificity. You can also use absolute positioning inside a relatively positioned element to place the search box precisely where you want it. The image will not scale with text resizing using just a single image. That would require a top and bottom image and what's known as the *sliding doors technique*—and that's another exercise for you!

Here is the CSS we must add to style the header:

```
#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;
   }
```

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

The header now looks as shown in Figure 9.9.


FIGURE 9.9 Header image background.

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

## Column Backgrounds

Recall that when you 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.

You have to use a technique called *sliding faux columns*, with which you essentially create two wide images that will slide over each other. You need to create two new containers to hold the backgrounds. Normally, you could apply one to the `#wrap div` that contains our entire page content, but here we use an extra (and wasteful) container for illustration purposes.

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

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

In this case, the maximum width is 960 pixels, so you start with an image of that width. In the image source files, it is `slidingcolumns.png`. You then export two slices (you can use the same slice and just hide/reveal the side images), one 960 pixels wide with a 192-pixel image for the column background on the left, and one 960 pixels wide with a 196-pixel image for the column background on the right.

**NOTE**
The left image needs to have a white background for the "long tail" to the right, and the right needs a transparent background for the "long tail" to the left. You can modify the color of the backgrounds as you export the images from the source file.

Where does 192 pixels come from? It's 20% of 960 (because the columns are 20% wide).

You use the `background-position` property to place the images in the correct place. Here, you 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 your `index.php` file, you simply add 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">
```

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

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

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

You must also put a background onto the 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;
   }
```

You need to clear the floats so that the float container (the faux columns) will extend to the bottom of the page. The best method for doing this is to use the property `:after` (see www.positioniseverything.net/easyclearing.html). But with the release of Internet Explorer 7, this method will not work completely.

A couple of solutions have been found (see www.quirksmode.org/css/clearing.html and www.sitepoint.com/blogs/2005/02/26/simple-clearing-of-floats/). You can use the *float (nearly) everything* option (see www.orderedlist.com/articles/clearing-floats-the-fne-method/).

Thus, you add a simple `clear:both` to the `#footer`, and you add floats to the `faux-col` wrappers.

```
/*Compass Design ie6only.css CSS*/
#leftfauxcol {
   float:left;
   width:100%;
   }
#rightfauxcol {
   float:left;
   width:100%;
```

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

**NOTE**
This won't work completely for IE6, but IE6 doesn't understand our min/max widths anyway and so will break the layout by trying to be fluid.

## Flexible Modules

Let's dig more deeply into adjusting the appearance of modules by setting display options and talk about the concept of chrome. *Chrome* is the nickname for giving content a stylized appearance, and Joomla has some built-in features that help you both eliminate tables from the layout and style in some shiny chrome. In your design, at the top position, you have a large initial module block. You don't know how tall the text that is needed will be. To solve that problem, you 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 you 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**
This code uses a conditional `else` statement to put in noncollapsing placeholder content, a nonbreaking space, so that if the top module location has no content, the full-size orange teaser image normally generated by the module will also not be there. What will be there is a nearly empty container that will contain a little of the background image and 20 pixels of vertical padding. This is purely for aesthetics.

The CSS needs to use CSS specificity for the `.top` module styles to override the generic `moduletable` styles defined earlier. These new styles specifically affect elements with the class `moduletable` that are descendants of the `div` with the `id` of `top` and would override any contrary settings for these style attributes set more generally for elements with the class `moduletable`:

```
#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 let's focus on some of the typography.

## Typography

Many of the links will need to be white, so you can 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 stylized button. You create this by 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 a list of FAQs, you can add an easy background by repeating the use of the image you used for the teaser:

**NOTE**
Reusing the image 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;
   }
```

The 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 style CSS. Here, you should keep it as simple as possible. You can slice a single image that includes both the bullet and the underline. Note that you turn on the styling by applying the module suffix `menu` to any menu modules that you 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, you 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 you use for our columns: the sliding doors again (see www.alistapart.com/articles/slidingdoors/)!

You can also try to do some speed optimization for the template and use just a single image for the left and right sides of the "doors," as well as the on and off states. This is known as using *sprites* (see www.fiftyfoureleven.com/weblog/web-development/css/doors-meet-sprites).

The CSS is not too difficult; you 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;
   }
```

You also need to add the module suffix `tabs` to the module for the menu you are using.

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

**NOTE**
Because the XML file for the template uses folders for CSS and images, you don't need to edit it to add these new files you have created because they will be automatically picked up and installed.

The finished template should look as shown Figure 9.10.

**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**
You now have a template based on a comp (or design). Some simple typography has been added, but more importantly, you 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`.

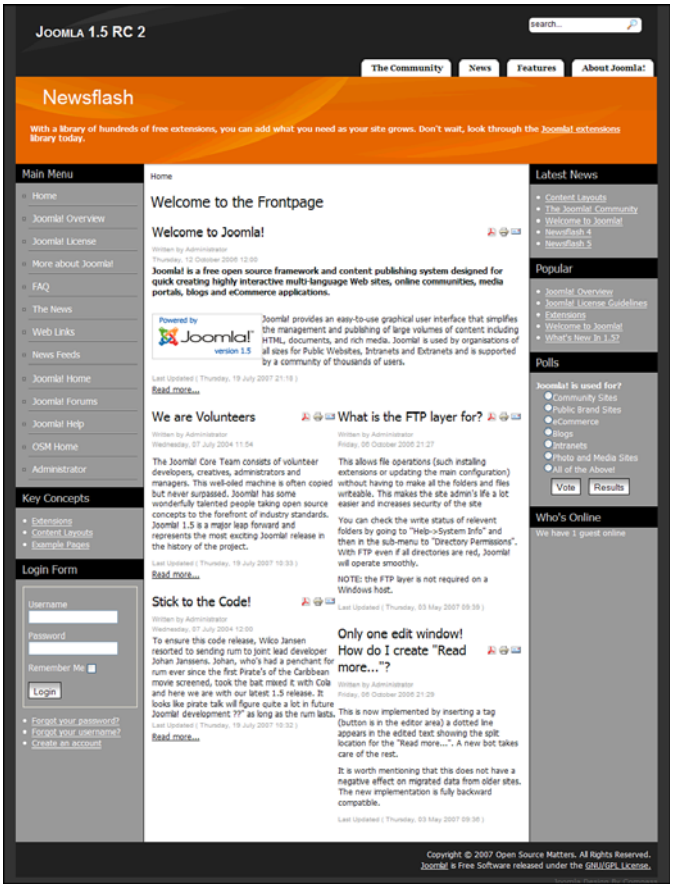


FIGURE 9.10 An advanced template with typography.

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

## Advanced Templating Features: CSSTemplateTutorialStep4

Joomla 1.5 offers a number of advanced template features that significantly expand what is possible with templates. You have already seen one example in this lesson: the ability to create custom chrome for modules.

Next, we'll examine template parameters and template overrides.

### Template Parameters

New in Joomla 1.5 is the addition of template parameters for templates. Template parameters allow you to pass variables to a template from options selected in the administrative backend.

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

```
<params>
<param name="template_width" type="list" default="1"
➥label="Template Width" description="Width style of the template">
   <option value="2">Fluid with maximum and minimum </option>
   <option value="1">Medium</option>
   <option value="0">Small</option>
  </param>
</params>
```

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

```
template_width=1
```

You need to make sure that this file is writable so changes can be made. When you use the Template Manager to open a template for editing, Joomla reports whether the parameters file is editable. If it isn't and your template needs to accept parameters, you need to change its permissions in the Cpanel. Unfortunately, there is a bug in Joomla that, based on certain conditions, causes Joomla to think the file is unwritable, when it actually is writable. You need to test this rather than rely on the interface text.

You also need to add `params.ini` as a file in the `templateDetails.xml` file list.

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

Based on the parameters you created above, you can see that it is a simple drop-down with three options. To display the options as radio buttons with a bit more detail about the choices, the param section of the `templateDetails.xml` file could look like this:

```
<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>
```

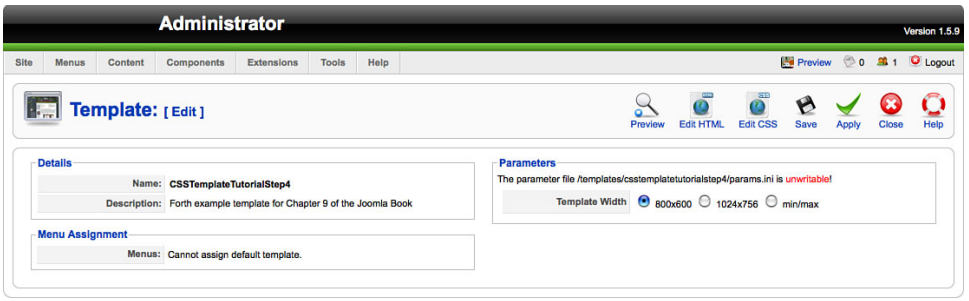


FIGURE 9.11 Template parameters in the administrative backend.

For the template width parameter to have an effect, you change the body tag in your `index.php` file to the following:

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

You then add the following to the CSS file, replacing the previous styles for `#wrap`:

```
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 you three options: a fixed narrow width, a fixed wide width, and a fluid version.

Using template parameters in this way gives you flexibility in almost any facet of a template—width, color, and so on—all controlled with conditional PHP being used to select from different CSS styles.

## Template Overrides

Perhaps the most powerful new feature of templates in Joomla 1.5 is the ability to easily override core output. You do this with new output files called *template files* that correspond to the layout views of each individual component and module. Joomla checks in each case to see whether an override file exists in the template folder, and if one is found, it uses the file instead of the normal built-in template for that module or component.

### Override Structure

All the layout views and templates in the main core are found in a `/tmpl/` folder. The location is slightly different for components than for modules because a module essentially has only one view. Here's an example:

```
modules/mod_newsflash/tmpl/
modules/mod_poll/tmpl/
components/com_user/views/login/tmpl/
components/com_user/views/register/tmpl/
components/com_content/views/article/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 a module has only one view.

TABLE 9.3 Examples of Overrides

| View | Layout | Templates |
|---|---|---|
| Category | `Blog.php` | `blog_item.php`<br>`blog_links.php` |
| Category | `default.php` | `default_items.php` |
| (Newsflash module) | `default.php`<br>`horz.php`<br>`vert.php` | `item.php` |

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

TABLE 9.4 Naming Convention for 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`<br>`blog_links.php` |
| `_templatename.php` | A common layout template used by different layouts | `_item.php` |

### Overriding Modules

Each module has a folder, called `tmpl`, that contains its templates. Inside it are PHP files that create the output. Here's an 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 examples 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. If you open that file, you find this:

```
<?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; ?>
```

You could change this to remove the tables and replace them with `<div>` tags to make the module's output a little more accessible and compliant with standards:

```
<?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
```

In general, you can copy a module's template file, edit it, and save it within the template. The path where you place the edited file for modules is *`templatename/html/modulename/samefilename`*. The advantage of saving into the template instead of just overwriting the core template file that's being edited is that doing so allows you to upgrade to successive versions of Joomla without fear that your carefully edited template revisions will be lost. Note that after installing an upgrade to the Joomla core, it is wise to check whether the core template files have been modified so that your custom versions of these files can be updated to match and take advantage of any new features.

You just took the tables out of the newsflash module—very easily!

### Component Overrides

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

If you look in the `com_content` folder, you see a folder called `views`, with a subfolder for each view:

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

These folders would match the four views for content: archive, article, category, and section. (Recall that Front Page is a special case that draws from the articles in the content component.)

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

If you look in the `category` folder, in addition to some XML files, you see this:

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

Note that in the case of `com_content`, the `default.php` layout generates the *list* layout, which presents articles as a list of linked titles.

If you open the `blog_item.php` file, you see the tables currently used. If you want to override the output, you put what you want to use in your `template/html/` folder. The path where you place the edited file for components is *`templatename/html/componentname/viewname/samefilename`*; here's an 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 for customizing your Joomla site through its template. You can create output templates that focus on SEO, accessibility, or the specific needs of almost any client.

**THE LEAST YOU NEED TO KNOW**

Joomla 1.5 offers new features for templates that allow designers to completely control the generated XHTML code and presentation of a Joomla website.

### Tableless Joomla!

The Joomla download contains a template called Beez that is a community-developed example of the template overrides in action. The Design and Accessibility team has 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.

To see a simple way to transfer the overrides from Beez to your template, watch this short video clip: screencast.com/t/Gka4ecnYr. This demo video depicts simply copying the entire `/html/` folder from the Beez template into your own template to achieve completely tableless layouts for core extensions. An advantage of this approach is that with each Joomla revision, the Beez template is modified with it. If you make no other changes to a specific component or to module template files, you can simply repeat the copy process to bring the latest Beez component and module template revisions into your template's `html` folder.

Notice, too, that as you add to your site third-party extensions that enhance any core functionality, you can utilize this override capability to implement their features as adaptations of core modules and components. As you install, ensure that your override files are not accidentally overwritten. You may need to edit the files to create a combined file, with code for both sets of features.

Recall that earlier in this chapter, we talked about the chrome options available in `modules.php` and the idea of adding your own options. As with overrides, the edited `modules.php` can be saved as `templates/`*`mytemplatename`*`/html/modules.php`. Be aware that third-party extensions also tend to add more chrome options to this file, so you may need to edit the file to combine the code segments for all the needed options into one file. (If installing a new extension ruins the display of previously working unrelated third-party modules, overwrites of overrides for this file is a likely suspect.)

**CSSTEMPLATETUTORIALSTEP4**

You now have a template based on a comp (or design). More visual typography has been added, but more importantly, you have used your pure CSS layout to create a template that has dynamic collapsible columns and a slick tabbed menu. You have also 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`.

## The Completed Template

Note that because the Joomla core's code is frequently changing and that override code needs to match the revision level of the software, the file `CSSTemplateTutorialStep4.zip` does not include the `html` folder with the overrides for a tableless layout. As described earlier, you need to copy the latest `html` folder from the latest Beez template into this template to complete it. Remember that to fully view this template's features

with your content, the module suffix needs to be changed for your various menus and that modules such as Newsflash have to be set to certain positions (for example, `top`, for which you have specified the applicable CSS).

## Summary

This lesson worked through four examples of templates, each time building the complexity and features. Here are the key points we looked at in this chapter:

- Modern websites separate content from presentation by using a technology known as Cascading Style Sheets (CSS). In Joomla, a template and its CSS files control the presentation of the content.
- When creating a template, it helps to have Joomla "running" on a localhost server so you can make changes at the file level with your favorite editor and refresh the page output in a browser to see the impact.
- Creating valid templates should be a path, not a goal. The idea is to make a 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, preferably in source order. Layout and design are part of the CSS, not part of the template.
- 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.
- Joomla will consistently 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.
- The output of modules can be completely customized, or you can use the prebuilt output options, such as `xhtml`. All these options are called *module chrome*.
- It's best to always use the fully expanded `list` options for menu output. You can then make use of many free resources on the web for the CSS that will style and animate your menu.
- 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 control whether any code associated with unused modules and their

container is included in the generated page; it is also done to link to different CSS styles to adjust the layout accordingly.

- 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 by revising the individual templates used by each component and module while still maintaining the upgradability of the core installation.

# Index

## C

## D



## E

## F



## G



## H

## M

## N

## O



## P



## Q-R

## S

## T

## U



## V-W

## X-Y-Z